1-1-2001

A scalable scheme for multilevel packet authentication in secure multicasting

Thomas Casey Reynolds
Iowa State University

A scalable scheme for multilevel packet authentication in secure multicasting

by

Thomas Casey Reynolds

A thesis submitted to the graduate faculty
in partial fulfillment of the requirements for the degree of
MASTER OF SCIENCE

Major: Computer Engineering
Major Professor: Manimaran Govindarasu

Iowa State University
Ames, Iowa
2001

Graduate College
Iowa State University

This is to certify that the Master's thesis of

Thomas Casey Reynolds

has met the thesis requirements of Iowa State University

Signatures have been redacted for privacy

TABLE OF CONTENTS

LIST OF FIGURES v
LIST OF TABLES vi
ABSTRACT vii
1. INTRODUCTION 1
1.1 Models of Communication 1
1.2 Multicasting 2
1.3 Multicast Security 3
2. ISSUES AND METRICS 6
2.1 Multicasting 6
2.2 Authentication 7
3. EXISTING MULTICAST AUTHENTICATION SCHEMES 10
3.1 Basic Methods 10
3.2 Multiple Keyed MAC 11
3.3 HMAC 11
3.4 UOWHFs 12
3.5 MMH 17
3.6 UMAC 19
3.7 On-Line/ Off-Line Digital Signatures 21
3.8 Multicast Solutions 24
3.9 Flow-Based Approach 27
3.10 Signing Digital Streams 30
3.11 Digital Signatures for Flows and Multicasts 33
3.12 Internet Protocol Security 36
4. PROPOSED SOLUTION 38
4.1 Existing Method 38
4.2 MeFFS 41
5. EXPERIMENTS 44
5.1 Experiment Setup 44

5.2 Experiments 47
6. CONCLUSIONS 55
REFERENCES 58

LIST OF FIGURES

Figure 1: Merkle-Damgard construction 15
Figure 2: Linear scheme 15
Figure 3: XOR linear scheme 16
Figure 4: Flow association mechanism 29
Figure 5: Star-chaining technique 34
Figure 6: Tree-chaining technique 36
Figure 7: Tree optimized for two levels of authentication 42
Figure 8: Randomly generated network 45
Figure 9: Multicast tree of Figure 8 45
Figure 10: Signature, packet size for new method 47
Figure 11: Signature, packet size for old method 47
Figure 12: Bytes sent per message for a 25 node network 48
Figure 13: Bytes sent per message for a 100 node network 49
Figure 14: Bytes sent per message for a 250 node network 49
Figure 15: 25 node, 4 level multicast network 51
Figure 16: Modified 25 node, 4 level multicast network 51
Figure 17: Bytes sent per message for 25 nodes 53
Figure 18: Bytes sent per message for 100 nodes 53
Figure 19: Bytes sent per message for 250 nodes 54

LIST OF TABLES

Table 1: Timing results for MMH implementation 19

ABSTRACT

Since communication has become an integral part of modern life, security and authentication have also become important issues for providing secure communication. In recent years, the issue of providing communication among a group of users has also been in the forefront, and multicasting has become a key technology for supporting such communication. This thesis researches the combination of multicasting and authentication, and proposes a new method for multicast authentication.

First, the thesis discusses several multicast authentication schemes: (i) two authentication methods, public-key systems and Message Authentication Codes (MACs), and their relation to multicast authentication, and (ii) different views of multicast authentication, including digital streams, off-line computation, and using less secure but more efficient authentication algorithms.

Then, the thesis proposes a scheme for multicast authentication based on a digital streams scheme, extended Fiege-Fiat-Shamir (eFFS), which is efficient and scalable. eFFS allows multiple levels of authentication, but sending these multiple levels may consume a lot of bandwidth. This research enhances eFFS (the enhanced scheme is called Modified eFFS, or MeFFS) by partitioning the receivers of a group to form several multicast subgroups, and sending different authentication levels to different subgroups. The performance enhancement of MeFFS, in terms of bandwidth savings, is quantified through simulation studies. Our studies show that MeFFS outperforms eFFS for multicast groups where the trees corresponding to the subgroups are highly disjoint. Finally, to make the scheme applicable to wide scenarios and to make the scheme scalable and easily deployable, some suggestions are made.

1. INTRODUCTION

During the past several years, the Internet has encountered an explosive growth, changing from a government and academic-only experiment to a tool used by much of the civilized world. Not only home users, but also businesses that are connecting have brought about this increase. As the Internet, its users, and its applications have expanded, so has the dependence upon the Internet's services. Networks are being used to send time sensitive information, such as stock trades and streaming audio/video. If this information is stopped, or even slowed, many companies would be seriously hurt by this degradation in service. Moreover, the data that both consumers and businesses send is becoming more sensitive and confidential, increasing the importance of the security of the data. Confidential business meetings, banking transactions, and credit card numbers are all being sent across the Internet; if this data is compromised, snooped, or delayed, users will become increasingly inconvenienced, if not seriously damaged. In addition to the sensitivity of the data, the complexity of the networks in which it needs to be sent through is rising, and handling Internet traffic is becoming increasingly complex. Packets are being sent to an exponentially rising number of nodes, and the number of packets generated by a single node may be increased due to the onset of broadcasting to a mass amount of users. This sends copies of the same packet to multiple users, which increases the bandwidth used by only one node, and can greatly congest segments of the Internet.

1.1 Models of Communication

The original forms of network communication were all unicast, or point-to-point. This occurs when user A communicates with user B, and user B responds directly to user A. In this paradigm, there is no option of speaking to a group of users, or an entire set of receivers. Broadcasting was then developed to better accommodate the needs of multiple communications. Broadcast communication is when a particular user sends data out to every user on its network. In this way, the host may send data to every user by sending only one packet. This is useful for some routing algorithms, which use flooding, and when a user needs to notify every node on the network, for example about an important notice. However,

if the communication is needed only to reach a small subset of the nodes, the network becomes congested as packets are flooded in the network. Moreover, all the non-intended nodes must filter out those packets, making the scheme inefficient, and possibly overloading some nodes. Various forms of group communication may be used to remedy this. The first solution is using many unicast communications. This will ensure sending packets only to the intended receivers; however, this involves sending a separate message for each receiver, which consumes a lot of bandwidth, thus making the scheme very inefficient. The next step is multicast communication. Multicasting is an efficient mechanism to send data to a group of users by sending a single copy of the data at the source and replicating the data, if required, at the intermediate routers between the sender and receivers. Multicasting is a highly bandwidth-efficient mechanism and has become the technology to support group communication over wide area networks such as the Internet.

1.2 Multicasting

Deering has proposed a definition for multicasting named the host group model [7]. This model consists of four basic components. The first component explains that a multicast group is formed by a group of users sharing a common address. In this way, a sender may send to one address and communicate with multiple users by sending only one packet. Moreover, the host group model stipulates that every receiver will receive the data sent to the group. This ensures that the senders are certain they will be sending to the complete group, not just a subset of it. The third piece of the model states that both group members and non-group members may send data. The address may be known by anyone, and users from within or outside of the group may send to the multicast address. There are some situations where this may not be desired, but in the general host group model this is a necessary feature. The last aspect of the model states that the senders do not know the membership of the group. Again, this may not always be a desired feature, but in general group communication the sender needs to only know the address it is sending to, not necessarily the particular users within that group. Multicasting is effective in conditions such as a site sending stock updates to multiple users, or an online video conference where each user needs to send to every other user, or a subset of those users. Instead of sending a separate message for each receiver, a

multicast address is used, and the intended receivers are assigned that address. Only one message is sent this way, saving the sender time of creating multiple packets with different addresses. In cases where several receivers are on the same network path, it also reduces bandwidth use by having only one copy of the packet being sent for all of the receivers on that path.

There are three main types of multicasting: One-to-Many (1toM), Many-to-Many (MtoM) and Many-to-One (Mto1) [20]. One-to-Many is when a single host is sending data to two or more receivers. Examples of this are scheduled audio/video distribution such as lectures, presentations, and television or radio broadcasts. Push media, such as news headlines and stock quotes, and file distribution are also examples of 1toM applications. Multimedia conferencing, such as audio/video and whiteboard meetings are classic examples of MtoM applications. In addition, distributed parallel processing, live distance learning, chat groups, and interactive multiplayer gaming are applications that use the MtoM paradigm. The last type, Mto1 is where any number of receivers send data back to a single source, either via unicast or via multicast. This can generate scaling problems, such as when multiple senders overwhelm the single receiver, known as an "implosion problem". Mto1 can be used for data collection, where multiple sensors or data collectors send data to a central collection point. Another example of Mto1 use is online auctioning; the auctioneer may send out the requested starting price, and the bidders send their bids back to the auctioneer.

1.3 Multicast Security

As the risks of having data corrupted or intercepted while traveling through the Internet is sharply rising, the importance of having much stronger security algorithms is also increasing. Instead of only personal or academic emails being sent across the Internet, company secrets, banking information, secure cellular phone calls, and much more sensitive information is being trusted to be sent securely from source to destination. To ensure high security and authentication, four major areas of multicast security need to be addressed [4, 25].

The first aspect is secrecy, or ensuring that only the intended recipient(s) may receive and understand data sent from various senders. There are three major components to network

and multicast secrecy. Ephemeral secrecy is the first component, which is preventing non-registered receivers from accessing group data. This is the basic component of secrecy – if a node is not a member of the group, then it should not be able to intelligibly read data sent to the group. This is accomplished with data encryption; there are many schemes currently available for this, and many more are being developed. Long-term secrecy protects data confidentiality for a long period of time. For example, if the data needs to be protected for a period of twenty years, using a scheme that is breakable in one month of computation time will not be sufficient. However, if the data to be protected are real time stock quotes, then the secrecy needs to last only fifteen minutes, because after that time any consumer may receive the delayed stock quotes free. The third component is join/leave secrecy. Join secrecy is not allowing a group member to read any of the data that was sent before it became a member. A good example of this is if a group is discussing whether to allow a particular member to join. The group does not want the new member to be able to see the previous discussions about the prospective member. Once the new member joins, the keys will be changed, encrypting the messages from that point on with a different key. Conversely, leave secrecy is when a particular member leaves the group, that node cannot read any of the group data from that point forward. In cases such as a business conference, leave secrecy is not an important factor. However, if a payment system is in place, once a user stops paying for access, that user should not have access for any further data; in this case, leave secrecy is important.

The second aspect of multicast security is anonymity, in which no one outside of the confidential group may determine who is a group member, or determine the sender of a particular message. In certain situations, the senders and receivers within a group may need to remain confidential. Traffic analysis is related to this facet of security; a third party should not be able to analyze traffic flows across a network to determine the senders and receivers of a particular secret flow.

Non-repudiation is somewhat the opposite of anonymity, and it deals with being able to prove that a sender did indeed send a particular message. Take the case of a transfer of funds from one bank account to another. If the sender of the funds later denies sending that money, non-repudiation is a method for the payee to prove to a third party that the message was indeed sent, and to be able to prove the sender of that data. This could also come into

play with contracts in court cases, when a party may deny signing a contract; with non-repudiation, the party will not be able to successfully deny that fact.

Authentication is the last major component of multicast security. This also is very important in unicasting; this thesis will focus on applying authentication to multicast networks, which will include methods originally developed for the unicast paradigm. Authentication may be divided into two separate functions: source and data authentication. Source authentication is being able to verify the sender of a message. If a user sends a message to another user asking him to respond back with sensitive or confidential data, the receiver of this message needs to be able to verify that he is indeed sending the data to an authorized person. Data authentication complements the source authentication issue. In the previous example, if the user responded with personal data, he wants to make sure that the data is not tampered with on its path to the receiver. Data authentication is the phase of security that will alert the receiver if anybody has changed even one bit of the message, allowing the sender to discard the message and not use corrupt data. Note that authentication is different from data encryption. Encryption is used in data secrecy, explained above, to keep non-intended receivers from successfully intercepting reading data. Authentication only makes sure that the messages have not been altered from sender to receiver. If only authentication is used, there is no guarantee that other users cannot read the data. This thesis is concerned with authentication methods, so will not always be concerned with secrecy. However, there are authentication schemes which use encryption, therefore providing both secrecy and authentication.

The remainder of the thesis is organized as follows. Section 2 will discuss the issues and metrics involved in effective multicasting and authentication schemes. Section 3 will introduce and explain several existing schemes for authentication. The proposed solution and the method it is improving is discussed in section 4. Section 5 will explain the experimental method, show the results, and introduce some improvements to the method proposed in this thesis. The conclusions are presented in section 6.

2. ISSUES AND METRICS

2.1 Multicasting

When several nodes join a particular group, there is more than one receiver to be concerned about, raising several issues. One issue is the member characteristics [4] – do all of the members have similar computing power, or at least a minimum of computing power? If the same packet or flow is sent to a high-powered machine as to a personal data assistant, the lower end machine will possibly be overpowered, and be forced to drop many packets. Therefore, a multicasting scheme should be able to accommodate various processing capabilities, and not be limited by one receiver's limited resources. Moreover, a multicast scheme should be efficient on any type of receiver. In scenarios where a server is sending to many users, then the scheme may require a powerful server. However, even in that case the algorithms used should be efficient for the receivers. When the scenario consists of similar peers communicating with each other, then the load needs to be evenly distributed. In this case, the processing time metric needs to be closely watched, and no group member should be unduly overloaded.

The routing algorithm used is also important in multicasting; the multicast trees will react differently when using sparse-mode or dense-mode routing algorithms, and the correct method needs to be chosen for each particular tree. Membership dynamics are a key factor in multicast groups [4]. If the group is highly dynamic and undergoing frequent joins and leaves, the tree must be constantly reconfigured, creating a high overhead. If this overhead is too high, it may cancel the advantage gained from using multicast instead of several unicasts. A particular multicast scheme may be better suited for a static group, where the members remain approximately the same, and joins or leaves are relatively rare. Whichever scheme is proposed for a given situation must be able to handle the group dynamics, whether they are highly dynamic or relatively static.

Scalability is becoming a principal concern for many issues relating to the Internet, due to its extreme growth rate. Multicasting and multicast security are no exceptions. A multicast system must be able to be vastly scalable; the desired size of multicast groups is only going to grow larger as more wide scale applications are developed. To accommodate

this, a scheme should be able to handle very large and distributed groups throughout the Internet. The beginnings of multicast are in local use and small group sizes, but the growth will lead to groups that are distributed globally, requiring a scheme that does not depend on localization.

As multicast groups become less localized, one additional factor will also become more important. This is the reliance of schemes upon network stability [19]. Local or metropolitan networks are relatively stable, not leaving a particular ISP. However, as the groups grow larger, the members will be located on varying degrees of reliable networks, and the multicast schemes cannot rely upon the networks to provide error free communication. This is especially important in many of the types of data that are used in multicast, such as audio and video streams. This data is sent over UDP, not TCP, which means that if a packet is lost or a router is overloaded, those packets will be discarded and no attempts will be made by the network to request another instance of that data. If a proposed scheme relies upon the network to deliver every packet and to guarantee ordered sending, once one packet is lost the scheme will fail. To be successful on the Internet and with large and distributed groups, a multicast plan must not be dependent upon networks to provide completely reliable service.

These are only some of the issues that must be addressed when creating multicast scenarios. If these issues are properly taken care of, multicasting may create a much more efficient method of exchanging data throughout the Internet.

2.2 Authentication

Authentication schemes have many of the same issues and concerns as multicasting. However, other issues and metrics come into play when authentication is added whether used in unicast networks, or in multicasting, as this thesis will address. One of these issues pertains to join/leave secrecy. If the secrecy is very important in a given situation, and the group membership is dynamic, then there will a large number of key changes to enforce the secrecy. In this case, or in any case where the overhead due to key changing must be kept to an absolute minimum, the authentication scheme must be able to gracefully and efficiently implement key updates.

A major concern of authentication schemes is the potential for collusion [19]. Collusion, as related to multicast security, occurs when two or more members of a group work together to attempt to defeat the authentication scheme. An algorithm may be able to tolerate up to N users colluding against it; however, once the $N+1^{th}$ user joins in the attack effort, then the scheme will fail. Because of this, the situation must be explored thoroughly before applying an authentication method that may be vulnerable to this type of attack. If the members of the group may be trusted, for example within a workplace environment, then collusion will not be a major issue. However, if the members are customers paying for a multicast service, then trust is diminished and there is an increased possibility of members attempting to gain free access or send falsely authenticated data. Depending on the given conditions, different schemes, or different levels of the same scheme must be used.

Time, along with bandwidth, is a major target of savings for multicasting. By sending only one packet to multiple receivers, less time and less bandwidth will be used. However, if each packet then needs to be authenticated, that counteracts the timesavings and, to a lesser extent, the bandwidth savings of multicasting. Therefore, the authentication and verification schemes both should be inexpensive to generate and verify. The additional information sent with each packet should also be kept to a minimum, to preserve the bandwidth savings. Another related factor is latency, which may be introduced by some authentication schemes. If several packets must be collected before authentication occurs, scenarios where data is bursty or must be sent instantly may be poorly affected. As with collusion and many other issues, the scenario must be studied before deciding how many packets, if any, must be collected before they are authenticated.

No matter how efficient, scalable, or latency free a scheme is, if it is highly vulnerable to attack, then the scheme is not very useful. Two types of attacks that may be brought against an authentication scheme; chosen message attack and known message attack [8]. A chosen message attack occurs when an adversary attempts to generate a signature of a particular message after getting signed messages of the adversary's choice. In this case, the user being attacked answers the attacker's queries, and responds with signed messages. In the second case, known message attack, the opponent tries to authenticate messages after receiving random signed messages. If, in either of the cases, the attacker is successful on one

particular message, this is defined as existential forgery. However, if an attacker can falsely authenticate any given message, then it has gained total forgery over the given scheme.

3. EXISTING MULTICAST AUTHENTICATION SCHEMES

Authentication in unicast and multicast networks has been a topic of research for several years. There are numerous existing schemes that address this issue; each has their own advantages and drawbacks. The following section will discuss the pros and cons of several authentication methods and how they may be applied to the multicast paradigm.

3.1 Basic Methods

There are two major approaches to authenticating data: public key signatures and message authentication codes (MACs) [4]. Public key schemes fit well to the multicast paradigm. To use them, the sender signs all data sent with its private key, which nobody else knows. The receivers then must all know the public key to unlock the data; this public key is known to anyone who asks for it. When the receivers unlock the message, they then know not only that the data has not changed, but also that the sender is indeed who it says it is. This is because the sender is the only user who may sign messages that will be unlocked by the public key, and no other user knows the private key to attempt to pose as the sender. The problem with this method is that public key signatures are computationally expensive, and the signatures tend to be rather long.

MACs are the second widely used option for authentication schemes. A MAC creates an output from a shared secret key and the message itself. This output, the signature of the packet, is then appended to the packet itself for transport across the network. MACs are generally faster to compute than public key signatures; they also require that every receiver have access to a shared key, k. This is different from the public key approach, where any user could have access to the key. In this case, the shared key must be securely transmitted to each of the receivers. Moreover, since there is only one k, any receiver may pose as a sender by signing a message of its own with k. The next section will outline a proposed solution for this problem.

3.2 Multiple Keyed MAC

The scheme proposed in [4] details a method to keep receiving members of the group from posing as senders, and to keep collusion to a minimum. Instead of generating only one MAC signature with one key, each receiver has a subset of keys owned by the sender. The sender then signs each message with every key, appending these MACs to the end of each packet. Each receiver then verifies every MAC within its own subset. This solves the problem of receivers posing as senders, because no receiver knows all of the keys, and no other receiver holds the same subset of keys. The receiver may be able to sign messages with several of the keys, but never enough that other receivers will be able to verify each of its own key subset. This also makes collusion harder to do successfully. This is because to send authenticated messages as a certified sender, enough receivers must work together to know every key in the sender's set. It may be possible to find enough keys to trick some receivers, but to completely break down the scheme enough receivers must collude to determine every key. A drawback to this method is that the sender must sign with every key, and each receiver must verify every signature corresponding to its subset of keys. Using less complex signatures can counteract this since multiple signatures are being used. Some schemes even use a single bit output for each MAC; even this will require a great amount of effort for an attacker to correctly guess each of the one bit outputs, if enough keys are being used.

3.3 HMAC

Hashing Message Authentication Code (HMAC) is a method of packet authentication using cryptographic functions, as described by RFC 2104 [15]. HMAC is a formal definition of keyed MACs, and will be outlined in this section. There are many major goals in the definition and creation of HMAC. The first is to use currently available hash functions that perform well and are freely available for public use. The point of HMAC is not to define how to hash the packets, but how to use various strengths of hashing methods to create an authentication signature for the packet. A related goal is to preserve the original performance of the hash function used, without adding significant overhead to the function to suit the HMAC purposes. HMAC should also use and handle keys in a simple way, which is out of

the scope of this thesis; key management for authentication and encryption is a major area of research by itself. Another main goal is to have in place a well-understood cryptographic analysis of the strength of the authentication scheme based on the hashing algorithm used. This is so one may know the effectiveness and relative strength of the HMAC scheme when different base hashing methods are used. The last major goal is to ensure that the underlying hash functions may be easily replaced. When a faster or more secure hash algorithm is found, this should be able to be easily swapped in place of the older hash algorithm with a minimum amount of effort.

The basic equation of HMAC is H(*K* XOR *opad*, H(*K* XOR *ipad*, *text*)). H is the hash function, *ipad* is 0x36 repeated several times, *opad* is 0x5C repeated several times, *K* is the key, and *text* is the text to be authenticated. There is also padding appended to *K* or *text* to ensure the values will work together properly. HMAC is a combination of XORing with some known pad values, and the key *K*. In addition, a hashing function is performed on the intermediate values to further scramble the results and make it as difficult as possible to return to the original value.

MD5 and SHA-1 are the two most popular hashing functions used to date. A hash function takes any input and returns a different value of fixed length, according to the function's specifications. MD5 returns a 16-byte output, and SHA-1 returns a 20-byte output. The key aspect of hash functions is that they take a particular input from a universal input, and return an output from a limited (e.g. 20-byte for SHA-1) alphabet set. The problem with hashes is that if there are more possible inputs than there are outputs, then there is a possibility of a collision. A collision occurs when more than one input to a hash results in an identical output. However, since 20 bytes is 2^160 different combinations, the chance of collision is very small. If several collisions are found, this could be used to break the authentication scheme, or even to find methods to falsely authenticate certain messages. The importance of these one-way functions will be described in the following section.

3.4 UOWHFs

The purpose of hashing functions is to try to create a function that will provide as close as possible to a non-reversible output to each particular input. However, even some

well-known and widely used attempts at this have failed – it has been recently proven that MD4 and MD5 can both be broken [1]. To overcome this, stronger algorithms such as RIPEMD-160 and SHA-1 have been created. These algorithms are more cryptographically secure, but they are slower as a result. One solution, other than attempting to create a perfect hash function, is to lower the standards for these types of algorithms, and make up for the lower standards in another manner. Universal One Way Hashing Functions (UOWHFs) are an attempt to do this; instead of creating more conservative, and therefore slower algorithms, these functions are used in a complementary scheme with other components. Existing methods for UOWHF are relatively slow, using general or algebraic assumptions. To improve upon this, [1] proposes a new construct of UOWHF using existing hashing algorithms such as MD5 or SHA-1, much as HMAC is described above.

The key point of UOWHF schemes is the difference between Any Collision Resistance (ACR) and Target Collision Resistance (TCR). Both of these schemes attempt to make it difficult, if not impossible, for an adversary to find a collision of more than one input to their respective hash functions. ACR is the standard concept of collision resistance. For the following examples, M is the input message; M' is a second, potentially colliding message; F is the hash function; f is the output of the hash function; and K is the key with which to run the hash function. The purpose of any hash function is to provide M' for a given M, without allowing any adversary to find a collision. The adversary's goal is to find a message M which will produce a known M' output. In ACR, the adversary chooses an M first. The adversary is then given K, and with this information attempts to produce M' such that it creates a collision with M when run on F. The function F can be informally defined as "any collision resistant" if the ratio of the running time to the probability of finding a collision is large.

The crux of TCR is that it forces the adversary to choose its second message, M', before it is given the key K to the hash function. In this manner, the adversary must commit to a point of collision before seeing K, which adds a very high degree of complexity to finding any collision. Because of this, an adversary cannot use any old collisions as starting points to new collisions, as long as K is refreshed at appropriate intervals.

Now that a method for creating a possibly more secure hashing function has been proposed, how can this method be successfully used in a secure application? The problem is

that the collision-resistance depends on K; K values must be introduced effectively and efficiently into the system. The most direct manner to achieve this is to key an existing hash function, such as SHA-1 or MD5. One method is to key the hashing method with its initial chaining value, IV. However, this only changes the problem to breaking that particular hashing function when it starts with a random, known IV. Another alternative is to use the envelope method. This uses the format $F(K||M||K)$, where || denotes concatenation. However, this may be broken using a version of Dobbertin's attack. This attack centers around the compression function, and uses the property that if for any C one can find M and M' such that compress($C||M$) = compress($C||M'$), then the compress function, and therefore the corresponding hash function has been broken. A more secure method is to spread the bits of K within the message M. In this way, the attacker must first determine which bits are K, and then attempt to run a successful attack.

There are many proposals that extend the last method explained, interleaving the message and key bits; these proposals break the messages into blocks of fixed length m. The space of possible message blocks is Σm, and the input message is viewed as $M = M_1 \ldots M_n$, where M_i is a subset of Σm for each $i = 1, \ldots, n$.

The first proposal places n hashing functions in sequence and each hash function receives as its input the message block M_i, the output from the previous hash function C_i, and the common key K. If the hashing function H is ACR, then so is this method. However, if H is TCR, then several MD iterations of H on this same point K may reveal the key, causing the method to possibly break down. See Figure 1 for an illustration of this method. To improve upon the security of the first method, several keys may be used in sequence, as shown in Figure 2. This provides for a completely TCR system, but at the cost of a much larger total key size. If the message M is broken into n sub-messages, then the total key size will be $n|K|$, where |K| is the size of K. This value may become very large if many message fragments are being used.

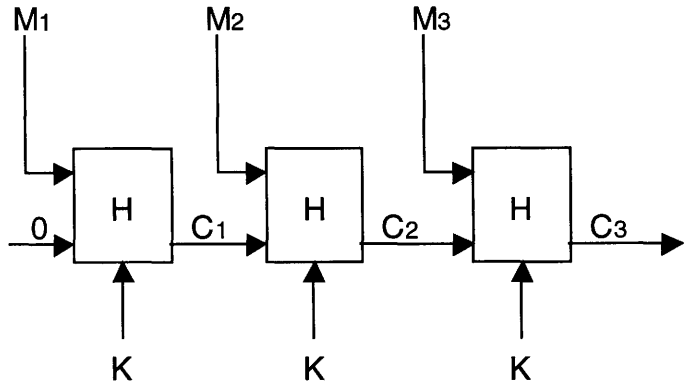

Figure 1: Merkle-Damgard construction

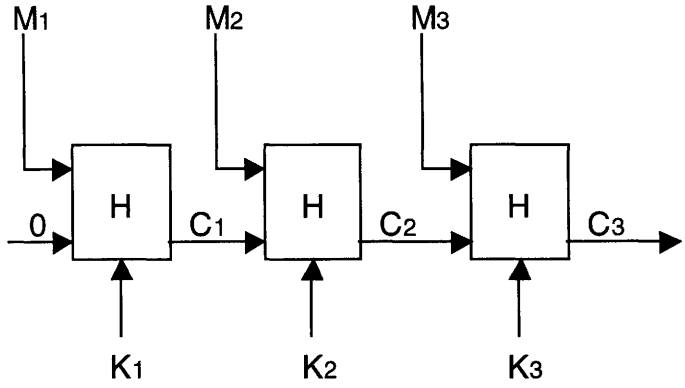

Figure 2: Linear scheme

To lessen the effect of many keys being used for a single message, XOR hashing has been proposed. The first method is XOR linear hashing, which uses the same key K in each iteration, but also introduces a masking key K_i, which is introduced between each iteration. The masking keys only need to be as large as the C_i outputs. In addition to saving key space, this also allows the same key to be used for each iteration, which will benefit hardware and software that is able to setup the key ahead of time. See Figure 3 for an illustration of this method.

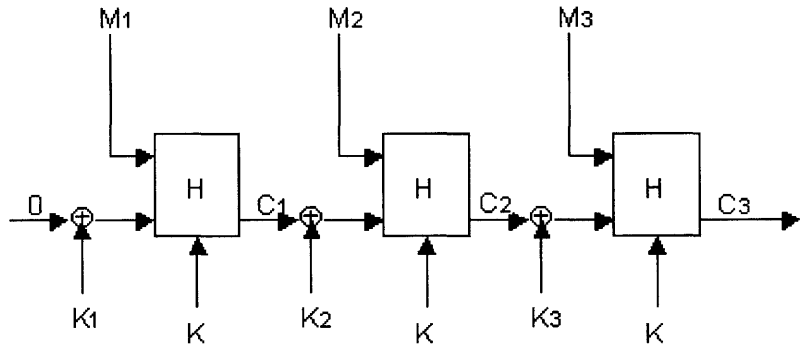

Figure 3: XOR linear scheme

To further reduce the key size necessary in a given scheme, a tree arrangement may be used in place of the linear schemes above. Also as above, a basic and an XOR version have been described. In the basic version of a *d*-ary tree, all of the hashes on one level of the tree are conducted simultaneously, which may also speed the hashing process. On the topmost level, *d* message blocks are put into each hashing function, along with key K_1. For example, if there are twenty-seven total message blocks, and three message blocks go into each tree, then there will be nine concurrent hashing functions. On the next level, these nine outputs will be placed, three at a time, into three other hashing functions, along with key K_2. The three outputs of these functions will be placed into one last hashing function, with K_3, and the final result will be this output. Not only does this save on the number of keys used, it also saves on the total processing time, when used on a system which can take advantage of parallel processing.

The last scheme, the XOR tree hash, adds the same type of auxiliary keys as XOR did to the linear hash. Instead of using a brand new key at each level, the same key is used for each hash function. However, the outputs of the hashing functions (three outputs in the above example) are all XORed with a new key K_i at each level *i*. This scheme is more complex than the others are, but it also can produce the fastest scheme, and one that uses the least amount of key bits while retaining a high level of collision resistance.

3.5 MMH

The purpose of the Multilinear-Modular-Hashing (MMH) proposal is to construct a very fast software implementation of almost universal hashing functions [10]. This takes current methods and applies several modifications to improve upon the performance to Gigabit-per-second hashing speeds on a 200 Mhz Pentium-Pro. Carter and Wegman first introduced universal hashing functions in 1979; as described in the above section, these functions have recently been applied to message authentication. Universal hash functions can be described as collections of hash functions that map messages into outputs, with a small probability of collisions between two messages. The collision probability in universal functions depends on the random choice of the hashing function used to hash the data. To use universal hashing for secure message authentication, the message is hashed by a universal hashing function; this output is then encrypted with a one-time pad. This result is sent with the original message so the receiver may attempt to perform the same functions and verify the message.

The key of universal hashing is to allow both the sending and receiving parties to know which hashing function from the universal hashing set is used for the particular communication. To accomplish this, both parties store a random index into a function within the universal hash family, along with the one-time pads used for encryption. In actual use, the random pads are of pseudorandom quality, and the encryption strength depends on the generation of these pads. There are two very beneficial qualities of this scheme. The first is being able to decouple the cryptographic functions from the majority of the work on the data. The second is taking advantage of the relaxed requirements of a universal hash function as opposed to a normal (ACR) hash function. More efficient constructs and methods may be used due to this relaxation of requirements.

To create efficient hashing schemes, the existing methods are implemented and then modified to remove their most costly operations. In [10], the modifications are geared toward new processor technologies, such as MMX extensions and 64-bit architectures. The goal is to create the fastest authentication scheme possible, while retaining high collision resistance. This is so a processor may spend the least amount of time possible on authentication schemes and be free to run its other operations.

The first definition, MMH*, comes from Carter and Wegman's original scheme. This definition works in the finite field Z_p for a prime integer p. The family of the hash functions consists of all of the multilinear functions over Z_p^k for an integer k. The family MMH* may be defined as:

$$MMH^* \stackrel{def}{=} \left\{ g_x : Z_p^k \rightarrow Z_p \mid x \in Z_p^k \right\}$$

The functions g_x are defined for any $x=(x_1, \ldots, x_k)$, $m=(m_1, \ldots, m_k)$, $x_i, m_i \in Z_p$:

$$g_x(m) = m \cdot x \bmod p = \sum_{i=1}^{k} m_i x_i \bmod p$$

To reduce the probability of collision to less than the above provided $1/p$, the message may be hashed twice. This will reduce the collision probability to $1/p^2$, but will double the computational work output, and will require twice the key size. If a Toeplitz matrix construction is used, then the key size may be reduced. Instead of selecting two independent vectors x_i through x_k, $k+1$ scalars x_1 through x_{k+1} are chosen; x is set to $(x_i, \ldots, x_k)$, and x' is set to $(x_2, \ldots, x_{k+1})$. This choice of keys will still result in a collision probability of $1/p^2$, but will only increase the key size by one scalar x_{k+1}.

The first major changes to the MMH construction are to use 32-bit integers and a prime integer $p=2^{32} + 15$. This allows the modified MMH (MMH^*_{32}) to take better advantage of 32 and 64-bit architectures, and the new prime number length allows a division-less version of modular reduction to be implemented. Modular reduction is the most expensive operation of MMH^*_{32}. Through arithmetic manipulation, the modular reductions of MMH may be reduced to much less expensive addition, subtraction, and comparison operations. This may still take 10-15 machine instructions; however, these will add up if executed frequently. To lessen the instruction count the inner-product method may be used. In this method, the entire inner-product operation is done over the integers, and the modulus operation is applied at the end of the other operations. This will be multiplying two 32-bit integers, so a machine with 64-bit operations is required, which many of today's processors can now handle.

One other modification to the base MMH method is to fix k, the length of the message and key vectors. When k is increased, the speed should also be increased, since the costly modular reduction is amortized over k multiply and add operations. However, the key

x consists of k 32-bit integers, so increasing k creates a larger key size. A value of k=32 provides a reasonable compromise, making modular reduction comprise only 10-15% of the total implementation cost.

The last modification described is only a direction to pursue, not an actual implementation. It explains that the structure of the hashing procedure, especially the inner-product calculations, is very conducive to parallel processing. Although this is left as future work, this insight shows that MMH has areas in which its performance can be improved.

Experiments were run on several types of processors [10]; this thesis will overview the fastest results, on a 200 MHz Pentium-Pro machine running Linux. For these experiments, MMH was hand-optimized in assembly, and the tree structure was coded in C. Two versions of MMH are used in this experiment: a basic and high security version. The basic method uses tree hashing and has 32-bit output, and a collision probability of $1.5/2^{30}$ times the height of the tree. The high security method uses the Toeplitz construction, creating a 64-bit output and a collision probability of $2.25/2^{60}$ times the height of the tree. Moreover, two different tests were run on the basic and high security methods. The first is when the message is long and is stored in a 4MB memory buffer, repeated 64 times. The second type, "message in cache", is when the data is accessed it always takes it from the same memory buffer, therefore always accessing the faster cache. Table 1 shows the results provided.

Table 1: Timing results for MMH implementation

200 MHz Pentium-Pro	*Message in memory*	*Message in cache*
64-bit output	380 Mbit/second	500 Mbit/second
32-bit output	645 Mbit/second	1080 Mbit/second

3.6 UMAC

Universal Message Authentication Code (UMAC) was created for two major reasons: extreme speed and provable security. The creators of UMAC set out to create the fastest MAC possible by a wide margin, while also being demonstrably secure. The previous fastest

algorithm is MMH, described above. UMAC does outperform MMH, as will be explained later in this section.

As with MMH, UMAC's starting point is the universal hash function family. More specifically, a set of universal hash functions is known as "ε-universal" if, for any pair of messages, the probability of the two messages colliding is at most ε. One major reason for using hashing before encryption is that hashing a message first provides a shorter message to encrypt. Therefore, if a more conservative encryption algorithm is desired, this will be applied to a shorter hash output of a message, rather than the message itself.

The new hash function family created for UMAC is called NH, and is a simplification of MMH. In NH, the message M is set into an even number of l integers, $M=(m_1, \ldots, m_l)$, where each $m_i \in \{0, \ldots, 2^w-1\}$ corresponds to a w-bit word, usually 16 or 32 bits. A particular function is named by a sequence of $n \geq 1$ w-bit integers $K=(k_1, \ldots, k_n)$. $\mathrm{NH}_k(M)$ is computed as: $\left(\sum_{i=1}^{l/2}((m_{2i-1}+k_{2i-1}) \bmod 2^w)\cdot((m_{2i}+k_{2i}) \bmod 2^w))\right) \bmod 2^w$

The best aspect of this equation is that it is computer-friendly arithmetic. There are no finite fields or non-trivial modular reductions. All of the modulus functions are to some power of two, making it very efficient to compute

Unlike many other MAC schemes, UMAC uses a stateful construction for the sender. To authenticate a message with a key, the sender must also provide a 64-bit nonce, which may not be reused. The output of this function is denoted as the authentication tag. The sender will need to send the receiver the message, this tag, and the nonce, so the receiver may compute the tag for itself to authenticate the message. A subkey generation process is also used with UMAC to convert from a shared, convenient length key to an internal key that is typically used for the duration of the communication session.

To further reduce the probability of collision, UMAC also allows for hashing the message twice. The keys used are shifts of each other, using the previously mentioned Toeplitz construction. UMAC and NH allow for multiple iterations of Toeplitz construction, which provides a method for easily increasing the security of the hashing scheme. UMAC also takes advantage of Single Instruction Multiple Data (SIMD) architectures by making small adjustments to the indexing of data. Instead of computing $(M_1 +_{16} K_1)$ by $(M_2 +_{16} K_2)$,

$(M_1 +_{16} K_1)$ by $(M_5 +_{16} K_5)$ is computed. This takes advantage of MMX instructions that treat 64-bit registers as four 16-bit sums or four 32-bit products. Indexing in this manner places the data perfectly in the registers, allowing larger numbers to be operated on at once.

The performance of UMAC is, as its goal states, faster than any method proposed before it. According to [2], the originally proposed MMH can run at 1.2 cycles/byte with a 2^{-30} chance of forgery. UMAC, on a 350 MHz Pentium II, gives a peak performance of 2.9 Gbits/sec (0.98 cycles/byte) with a 2^{-60} chance of forgery. At a 2^{-30} chance of forgery, UMAC runs as fast as 5.6 Gbits/sec (0.51 cycles/byte). A standard SHA-1 implementation runs at 12.6 cycles/byte. These results show that UMAC can provide secure authentication at a very fast rate, allowing the host processor ample time to compute its other necessary operations.

3.7 On-Line/ Off-Line Digital Signatures

In the basic structure of an asymmetric digital signature scheme, each user U publishes its public key and keeps its own secret key. When U signs its message m, it creates a value σ. It is essential that U can quickly generate σ and any receiver can quickly verify σ's validity, using U's public key. It is also essential to make it extremely difficult for potential adversaries to forge U's signatures without knowledge of U's secret (private) key. What the on-line/off-line signature aspect adds is the notion of processing potentially slower precomputations before the actual message authentication. These off-line computations can be done independently of the message being signed; the second, on-line phase is computed when the message arrives. The general construction of the on-line/off-line signature scheme contains three main aspects. These aspects are as follows:

1. An ordinary signature scheme
2. A fast one-time signature scheme
3. A fast collision-free hashing scheme

The ordinary scheme is used to create the off-line signature of a randomly constructed instance of information that enables the one-time signature. It is also used to compute an on-line signature of the actual message; this signature is typically very fast. This is because the only computations required during the on-line phase are applications of DES; no modular

multiplication is required. The expensive modular computation of extracting square roots is calculated off-line, and will be ready to use when a message arrives. One-time signature schemes are much faster than ordinary schemes; however, the signing key may be used only once, and then a new key must be generated. In the on-line/ off-line scheme, the key only needs to be used one time, so this drawback does not affect the performance of the system. The last aspect, the collision-free hashing scheme, is mainly used to compress longer messages into shorter, more manageable blocks for faster signing times. The following paragraphs will explain in detail the three components, and introduce an additional aspect: using one-way functions for the one-time signatures.

The common definition of a signature scheme is a triplet (G, S, V) of probabilistic polynomial-time algorithms satisfying the following conventions [8]. Algorithm G is the key generator. G produces an output pair (sk, vk), where sk is the signing key, and vk is the verification key. Algorithm S is the signing algorithm. S takes an input (sk, M), where M is the input message, and outputs the signature of M. Algorithm V is the verification algorithm. The receiver computes V(M, vk, σ), and checks that the result is one, meaning the message is verified correctly. Note that these algorithms are composed to input and output messages of predetermined length. Padding shorter strings or hashing longer strings to the desirable length can adjust messages of other lengths.

A one-time signature can be defined as a digital signature scheme that can be used to legitimately sign a single message. This scheme is secure against known message attacks if it is secure against attacks that are restricted to a single query. This is one major benefit of one-time signatures. Since a new signature instance will be used for every message, an adversary has only one signature with which to attempt a break of the scheme. This is similar to a one-time pad, where a user draws from a particular pad of data for encryption, then throws the pad away and uses a new one for the next message.

Let (g, s, v) denote a one-time signature scheme, and let n denote the security parameter. This security parameter determines the lengths of the keys and messages of ordinary and one-time signature schemes. The key generation algorithm g is the same as the ordinary key generation algorithm G. During the off-line phase, a pair of one-time signing and verifying keys is created, and the one-time key is signed with an ordinary signature

scheme. Both the one-time key and its signature are stored for use during the on-line phase. In more detail, the signer runs its key generation algorithm g to randomly select a one-time verification key vk, and a one-time signing key sk. The signature of vk is then computed, using an ordinary signature algorithm S with its associated signing key SK. This can be symbolized as

$$\sum \overset{def}{=} S_{sk}(vk).$$

The signer stores (sk, vk) and the off-line signature Σ. During the on-line phase, a pair (sk, vk) of keys is selected from memory, and the message M is signed using the one-time signature key sk, creating the one-time signature σ. This can be symbolized as

$$\sigma \overset{def}{=} s_{sk}(M)$$

The signature of M consists of the triplet (vk, Σ, σ). To verify this signature, the receiver uses the verification algorithm V to ensure that Σ is a valid signature of the one-time verification key vk, with respect to the ordinary verification key VK. Next, the receiver ensures that σ is a signature of the message M with respect to vk. This is shown as

$$V_{vk}(vk, \Sigma) \wedge v_{vk}(M, \sigma)$$

The next aspect of on-line/off-line signatures is adding one-way functions to the signatures, and increasing the efficiency and security of the overall scheme. The basic construction starts with f, a one-way function that is assumed to be polynomial-time computable but infeasible to invert with any notable probability. The signing key is composed of a sequence of m pairs of n-bit long strings $(x_1^0, x_1^1),\ldots,(x_m^0, x_m^1)$, and applying the one-way function f to each of the 2m strings creates the verification key. To sign the message $\sigma_1, \ldots, \sigma_m$, the signer reveals $x_1^{\sigma_1}, \ldots, x_t^{\sigma_m}$, and the receiver applies f to the revealed strings. This output is then checked to see if they match the corresponding strings in the verification key.

This method is very secure; however, it does have drawbacks to be addressed. The first drawback is that very long keys and signatures are created in the basic construction. To shorten these, an idea attributed to Winteritz may be used. In this scheme, only m+1 strings are used, each of length n, while the basic construction uses 2m strings. A similar method of

signing is used in the Winteritz construction, except the signer reveals some slightly different data, and the verifier applies a somewhat modified function to the revealed data. To increase the security of this scheme, the use of error-correcting codes may be added. A message is first input to an error-correcting algorithm, and the resulting output is signed, instead of the original message M. An error-correcting code produces an output that is different by the input M from at least a Hamming distance of d. Due to this, not only does an adversary need to forge a signature to a string different from the original message, it must do this to a string different from the original message by at least a distance d. This enhancement, along with the previously mentioned efficiency enhancement, make one-time signatures an ideal use for the on-line/ off-line method, and help the on-line/ off-line scheme become a viable scheme for fast, secure message authentication

3.8 Multicast Solutions

Most of the above schemes were originally proposed for unicast solutions. They may be applied to multicast solutions with varying amounts of work, but most were created with unicast solutions in mind. This section will outline several solutions for multicast methods, and will explain one multicast method that is a derivative of the above on-line/off-line signature scheme. The basic unicast method is to sign each message with a MAC, as explained earlier. However, in a multicast setting, all users must hold the key, and therefore anybody holding the keys may read and/or write to the group. A second method is to sign each packet individually. This is a costly method, and in a multicast setting, a server may be signing packets for multiple groups, creating an even higher burden on the server.

One manner to address these problems is to relax the security requirements of the particular scheme. If a system does not require the highest, most conservative security methods possible, then it is acceptable to use a faster, less secure scheme. The obvious drawback is that if the method used is broken, then all messages sent will be open to attack until another method is implemented. Asymmetric MACs were then introduced, where a sender holds several keys for signing its messages. The keys are then distributed to the potential receivers in such a manner that if no more than w receivers collude, the scheme cannot be broken. For example, if the sender signs the message with w+1 separate keys, then

even if w members of the group shared their keys with each other, the message will still be secure. The problem of collusion arises when one additional member joins the collusion. If w+1 members work together, and they happen to hold the w+1 keys the sender used to sign its message, then the scheme will break down completely. One additional drawback is that the use of several keys creates a longer signature length. Schemes have been introduced to use 1-bit MACs for each signature, but these obviously have at least a somewhat reduced level of security.

The next potential method is digital streams, which will be described in the following section. The basic concept is that a stream is broken up into several parts, and each part is dependent upon an adjacent block of the stream. This is done by embedding the signature of one block of the stream into a different block of the stream. The major drawback of this method is that it requires a fully reliable network. If one packet does not arrive, then the entire scheme breaks down. Similar to this is dividing a stream into several blocks and signing them individually. A fully reliable network is not required, but if a large delay is necessary to collect the proper number of packets, this will not suffice for real-time applications. Moreover, if a server is required to sign multiple flows at once, keeping track of the blocks for each flow will greatly reduce the performance of the scheme, making this option impractical.

The solution introduced in [21] provides the proper security guarantees while working in an unreliable network and remaining efficient in both size and speed. This approach emulates the public key signature-per-packet approach while adding a derivative of the on-line/ off-line approach to increase performance. The major addition of this approach is to have the off-line component of the method create and certify k-time key pairs instead of 1-time key pairs. In this way, the certificate creation cost may be amortized over k signatures, making the average creation cost become possibly much lower. As the value of k increases, the sustainable rate will approach the potential signature rate possible from the k-time key pair generation processes. When k decreases, the signature rate will move closer to k times the regular signature rate. At the time of publishing of [21], the potential rate in practice was up to 500-1000 signatures/second.

The basic flow of this method is to have the sender create a process that generates k-time key pairs. This process also creates certificates for the k-time key pairs using the sender's long-term signature key. The sender then uses one of these key pairs to sign k successive messages. The certificate for the k-time public key is also sent to the receivers. To address varying levels of network reliability, this certificate may be sent with each packet in a very unreliable network or only once in a fully reliable network. The frequency at which certificates are sent can be easily tuned to a network's level of reliability. This method still has one major drawback that has kept it from widespread use. This drawback is identical to that of the on-line/ off-line signature scheme: k-time public key signatures are very large and therefore impractical for many networks. The method described above, when used with a secure k-time signature scheme, can produce signature sizes of up to a kilobyte per packet, which is obviously unreasonable.

This method improves upon this drawback with three adjustments.

1. Using TCR hash functions to reduce size overhead
2. Use known method to reduce ancillary authentication information
3. Reduce certificate size with previously known methods

With these improvements, packet overhead can be reduced to less than 300 bytes.

The first improvement uses a previously discussed method, Target Collision Resistant hash functions, also known as Universal One Way Hash Functions. Instead of using one collision resistant hash function to reduce the message size, a family of weaker, keyed hash functions is used to hash the messages. The signer first chooses a hash function at random, computes the hash value, and then signs the hash value and the associated key. TCR can reduce the size of the hash of the message to be signed, but this is not sufficient because the random key used to choose the hashing function must also be signed. The best solution would be to have the sender sign the hash of the message and not the key. However, due to the definition of TCR, the signer must choose the key only after the message has been chosen, and the key must also be authenticated. To solve this problem, commitments to the keys used to select the TCR hash function are included in the regular signature of the k-time public key. For each message, only the TCR hash value is signed with one of the k-time signatures, and the key already committed to will simultaneously be revealed. Using this

method, the security of TCR is kept as normal, the message signatures are still almost halved, and the speed of the underlying k-time scheme is almost doubled. If necessary, the speed improvements gained by using TCR commitments can be traded for space improvements, reducing the original space overhead of the k-time signature scheme by three.

The next improvement affects the ancillary information; this information, contained within a k-time signature, identifies with which one of the k uses this particular message is signed. In general, a k-time public key is created from k one-time public keys via a collision resistant hash tree construction using the k public keys. The root of this hash tree is the k-time public key. When a message assigned with the i^{th} one-time key, this signature must also include the i^{th} public key and hashes of the sibling nodes on the path from that public key up to the root. This method replaces the collision resistant hash tree with a TCR hash tree. This replacement halves the sizes of all of the interior nodes of the tree, with one TCR key used for each level of the tree. These level keys are signed in the certificate for the k-time public key, so they will not need to be sent with each signature. This also halves the size overhead for the path from the i^{th} signature to the root of the tree. The one drawback to this is that the security is somewhat weaker due to the use of less conservative TCR hashing. To address this, the commitments will be termed "TCR-commitments". The sender may later change these commitments, but it has no incentive to do so, making this drawback acceptable.

The last optimization reduces the certificate size by using a currently known padding scheme. The k-time public key is embedded inside the regular signature within the certificate itself. This saves on the overhead of sending both the k-time public key, and its certificate. This improvement, along with the other described improvements, allows the k-time on-line/ off-line signature scheme to be a viable and efficient solution for authenticating packets in a multicast network.

3.9 Flow-Based Approach

The next method to be introduced is a flow-based approach to authenticating information. The crux of this idea is to use flows as the basis for signatures. A flow can be defined as a sequence of datagrams satisfying some pre-defined attributes [18]. A flow is neither a datagram nor a connection-based method; instead, it combines the best of both

ideas. This section will describe both previous methods, including datagrams and connection-based sessions, and explain a new method using flows as its base component.

A datagram service is one where self-contained messages (datagrams) are transmitted from one principal to another. In this context, a principal can be a host, a process, or a user. Each datagram is sent and received between principals atomically and independently, with no previous setup required. This concept is used by several widely used protocols, such as IP, UDP, and RPC. Many protocols have been recently introduced to attempt to secure network communications based on datagram services. However, these proposals still have several drawbacks. Several methods adopt a session-based model for adding security, such as requiring additional message exchanges for secure communication, and the use of hard states. The use of session-based security does create a well-defined unit of protection, but there is still a loss of datagram semantics. Many proposals also focus on specific protocol layers, instead of creating a general solution that may be applied to varied layers or processes. Lastly, existing protocols concentrate on specific mechanisms while not addressing policy issues. Most methods may be deemed useless if their associated policies are not implemented correctly; therefore, any practical method must be amenable to policy controls.

The current proposal is the Flow-Based Security Protocol (FBS). This protocol is based on the concept of flows, and makes use of zero-message keying and soft states. The former allows FBS to maintain datagram semantics, and the latter makes its efficiency comparable to connection-based approaches. FBS adheres to several general and self-imposed requirements and constraints. The first requirement is that it provides a prescribed set of security properties. In this method, the set enforces separation between datagrams belonging to different flows. FBS should also preserve datagram semantics; this means that neither party should require state setup messages or hard state maintenance. The last constraint is that the design should be layer and protocol independent. Creating an abstract protocol, then defining specifics for mapping to a particular layer or protocol, allows FBS to adhere to this constraint.

The most difficult challenge in a flow-based security approach is flow identification. To accomplish this, FBS has created the flow association mechanism (FAM). This mechanism separates outgoing datagrams into flows, while the zero-message keying

mechanism establishes security parameters for a flow without sustaining additional end-to-end message exchange. These two mechanisms work closely together to create a flow identifier named the security flow label (sfl). The sfl is then input into the zero-message keying mechanism to produce the per-flow key. To create a policy-driven FAM, several policy modules are used that plug into the FAM, as shown in Figure 4.

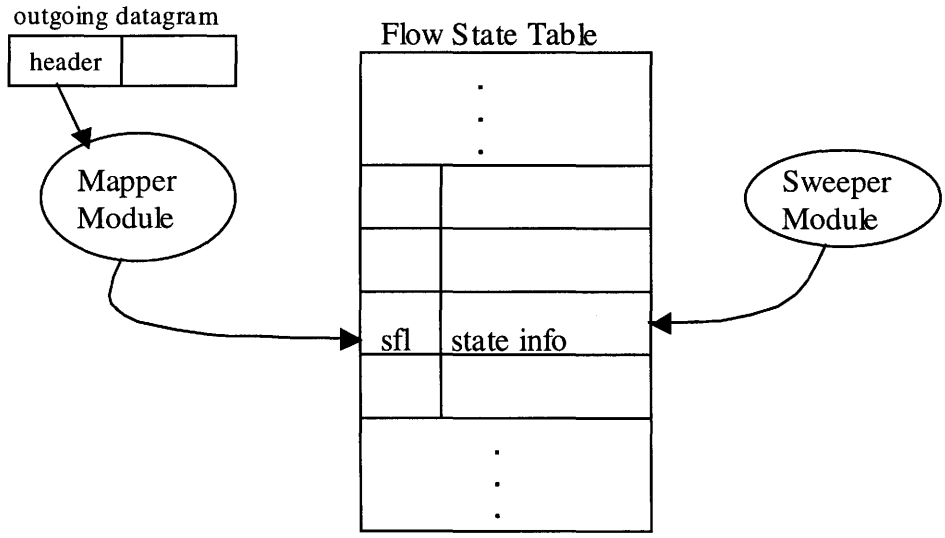

Figure 4: Flow association mechanism

The above design contains three major elements. The first is the flow state table, which stores information about each active flow. Each entry stores the sfl for a flow, along with state information needed by the mapper and sweeper modules. The mapper module takes as input a set of attributes of a datagram and potentially additional system parameters. The module produces an index into the flow state table with these inputs. A valid output is determined by inspecting its state information. If an output is valid, the datagram is deemed to belong to the corresponding flow; if it is invalid, a new flow with a new sfl is created and used. The sweeper module's duty is to remove flows no longer active. The module searches through the state table and determines whether or not to expire a flow by examining its state information.

A sfl is added to each datagram sent using the FBS method. In addition to this sfl, three other pieces of header information are prepended to each datagram. The first additional

data is the confounder, a statistically random value generated for use in the computation of the MAC and as the initialization vector for the MAC. The confounder aids in masking identical datagrams within a flow; if not masked, these datagrams may help an attacker in breaking the authentication scheme. The next header segment is a keyed MAC for each datagram. This MAC is keyed with the flow key and is calculated over the confounder, timestamp, and payload. The MAC ensures the integrity of the datagram from changes by third parties; it also provides a form of flow authentication, ensuring the datagram belongs to the flow denoted by the sfl. The MAC in FBS is defined as

$$\text{HMAC}(K_f \| \text{ confounder} \| \text{ timestamp} \| \text{ payload}),$$

where HMAC is a one-way cryptographic hash function. The last piece of the header, the timestamp, serves as a value for countering replay attacks. If the receiver sees a message with an identical timestamp, it will know to discard this message.

Zero-message keying is accomplished using the Diffie-Hellman (DH) key exchange protocol. DH is used to create a master key, and a flow key is derived from the former key and the sfl. The flow key cannot be used to determine the master key, making this aspect secure. By including the sfl in the datagram header, a destination principal may calculate the flow key without needing any extra principal-to-principal message exchange. Moreover, if the flow key is cached at both principals as a soft state, the flow key calculation may be amortized over all datagrams sent in that flow. All of the above characteristics provide a viable method for separating datagrams into flows, and providing a method for efficiently authenticating these flows via a header added to each datagram.

3.10 Signing Digital Streams

The concept of signing digital streams has been briefly introduced in this thesis. A stream can be defined as a potentially very long (infinite) sequence of bits that a sender sends to a receiver [9]. The key aspect of a stream is that the receiver must be able to process the data it receives at approximately the input rate. This means the receiver does not have the luxury of buffering large amounts of data; it must take the data as received and process it as soon as possible. Examples of these digital streams are live audio and video feeds, data feeds, and applets. Each of these is sent as either a finite or an infinite stream of data.

Several methods have been previously proposed to handle the authentication of digital streams. One of these methods splits the stream into blocks. The sender signs each block; the receiver then loads an entire block and verifies its signature. This solution works for an infinite stream, but the sender is required to generate a signature for each block sent. Signature generation and verification can be expensive, making this method a potentially too expensive to be practical.

One other proposed method works only for finite schemes; this method creates a table listing the hashes of each of the blocks. The sender signs this hash table, sends it to the receiver, and then sends the stream itself. The receiver then stores the table, verifies its signature, and then compares the hashes stored in the table to the hashes of each of the blocks. If each hash is a match, then the entire stream is verified. The major drawback of this solution is if a very large stream is being sent, then the table may also be large, creating storage problems and delays while signing the table. It is possible to split the stream into several sub-streams, where each sub-stream's table is then signed and sent to the receiver. Moreover, these blocks may be placed as leaves of a binary tree. Each node takes the hash of its children, and the sender only needs to sign the root of the tree. However, to authenticate each block, the sender must send the entire authentication path to the receiver.

The solution presented in [9] starts with some assumptions about the environment and participants. It is assumed it is possible for the sender to embed authentication information into the stream, and the receiver has a small buffer where it can authenticate received bits processing them. The receiver is also assumed to be able to authenticate fast cryptographic checksums faster than the incoming stream rate, while being able to play the incoming stream in real-time. With these assumptions, the stream is divided into blocks, and authentication information is embedded into the stream. The information from the i^{th} block is used to authenticate the $(i+1)^{st}$ block. The signer then only needs to sign the first block; this signature will propagate through the rest of the stream via the authentication information.

Digital streams can be associated in two basic scenarios. The first is a finite stream, where the sender knows the entire stream in advance. This is the case when the stream is a movie broadcast, or a pre-recorded program. The stream is divided into blocks of size c, and the receiver has a buffer of size c available to receive the stream. When the receiver is sent

the signature of the 20-byte hash of the first block, it verifies that signature. The receiver now knows what the hash of the first block should be, verifies that with the first block, and uses the hash sent in the first block to verify the second block. This process is repeated until the final block is received. To explain in more detail, let the first block be denoted as $B= B_1, B_2, \ldots, B_k$. The signed stream may be denoted as $B'= B'_0, B'_1, \ldots, B'_k$. The processing of the original stream is done backwards. The final term, B'_k, is denoted as $<B_k, 00\ldots0>$. Any B'_I, for $I= 1, \ldots, k-1$ can be described as $<B_i, H(B'_{i+1})>$. This means that any element of B' is the current block, plus the hash of the next block in the stream. Lastly, B'_0 can be denoted by $<H(B'_I, k), S(SK, H(B'_1,k))>$. B'_0 is a sequence of the hash of the first block with respect to k, and the signature of the sender's secret key and the previously mentioned hash. This allows the receiver to verify that the hash of the first block is valid and to use that to verify the first block, and accordingly the rest of the stream.

The second scenario is when the stream is not known in advance, such as a live video stream or broadcast. In this scenario, the signing process must also be fast, because the sender will be signing the stream in real-time, and cannot do it beforehand. The stream is also split into blocks in this scenario. The sender first sends a signed public key for a one-time signature scheme. The one-time signature scheme is used for its speed, which has been discussed previously in this thesis. After the key is sent, the sender sends the first block along with a one-time signature of its hash, based on the one-time public key sent in the previous block. The first block also holds a new one-time pubic key that will be used to verify the signature of the second block. This process repeats until the last block of the stream is encountered. In more detail, let the stream B be a sequence of blocks $B= B_1, B_2, \ldots$. This stream has no ending block, because the end of the stream is not known as the sender is processing it. Let $B'_0= <pk_0, S(SK, pk_0)>$ where pk is the sender's private key and SK is the sender's long-lived signature key. Each following block B'_i is then sent as $<B_i, pk_i, s(sk_{i-1}, H(B_i, pk_i))>$ for $i\geq1$. In these blocks, the block itself is sent, followed with its public key. The next element of the block is a one-time signature, with respect to the previous block's secret key, of a hash of the current block with respect to the current public key. The receiver then uses the key sent in each block in the stream to verify the next block of the stream, until the stream is finished.

There are several variations of the above two scenarios, using different algorithms for hashing and signatures. With the proper algorithms, this system provides an efficient solution for verifying data streams, whether they are known a priori or not. However, the major drawback of needing a fully reliable network remains in the most efficient solutions. If this requirement can be met, this method is very viable; unfortunately, many of today's networks cannot provide this level of performance.

3.11 Digital Signatures for Flows and Multicasts

The basis of the method proposed in [25] follows several characteristics in the delivery of flows within an unreliable packet network. These characteristics are as follows:

- Each packet in the flow should be used immediately upon arrival.
- A receiver may not get every packet within a flow; not every receiver will get the same subsequence of packets.
- Time-sensitive flows require fast input processing at the receivers. Real-time flows also require fast processing at the senders.
- In a multicast scenario, some receivers may be of limited resources, such as personal data assistants, or older computers, when compared to the sender, which is usually of server quality.
- Some receivers may be slow, but some may also be fast; this range may be wide.

Using these characteristics, a system was designed that meets the following requirements:

- The signing procedure is efficient, and delay-bounded when necessary.
- The verification procedure is also efficient.
- Packets within a flow may be individually verifiable.
- Packet signatures are small, creating a small communication overhead.
- The verification level is adjustable to the processing level of the receiver, allowing a slower receiver to return after the fact and further verify packets.

To accomplish these requirements, two chaining schemes have been proposed to sign and verify multiple packets, denoted as a block, in a single operation. Since all of the packets

in the block are signed and verified in only one operation, the procedure can be amortized over the entire block, making the overall rates much faster than other proposed methods. The basic function of both of these methods is to compute a block digest of each block, and then have each packet also contain authentication information. This authentication information consists of the signed digest of the block and some addition chaining information so the receiver may verify where it belongs in the block.

The first chaining method is denoted star chaining. In this technique, the block digest is the message digest of the m packet digests, where m packets are contained in one block. Let h() denote the message digest being used, and eight packets are in a block for this example. The block digest is the hash of the packets, or $D_{1\text{-}8} = h(D_1, \ldots, D_8)$. The block signature is sign($D_{1\text{-}8}$); this is the block digest signed by a digital signature algorithm. To relate the packet digest and the block digest, a one-level rooted tree can be created, and denoted as an authentication star. Figure 5 illustrates the authentication star.

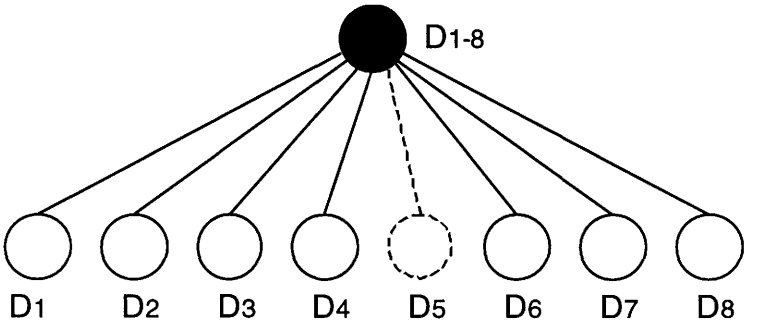

Figure 5: Star-chaining technique

To individually verify each packet, every packet needs to have its own authentication information, called a packet signature. This signature consists of the block signature, the position of the packet in the block, and the digests of the other packets in the block. As an example, let the fifth packet in Figure 5 be verified. The receiver first calculates the digest D'_5 of the packet, and then calculates $D'_{1\text{-}8} = h(D_1, D_2, D_3, D_4, D'_5, D_6, D_7, D_8)$. The signature of the packet received must also contain D_1, D_2, D_3, D_4, D_6, D_7, and D_8 for this to function correctly. The receiver then determines if the digest $D'_{1\text{-}8}$ is equal to the signed digest of $D_{1\text{-}8}$. If these two values are identical, then the packet is verified. Notice that the receiver has

received the signed values for all other packets in the block. If another packet in the block arrives, the receiver must only compare the stored digest with the value it calculates of the new packet.

The next method proposed is termed tree chaining, and it is a special subset of star chaining. In this method, the block digest is calculated as the root node of the authentication tree. As an example, see Figure 6, where eight packets are organized in a binary tree. The packet digests are the leaf nodes of the tree, and the other nodes are the message digests of their respective children. The parent of leaves D_5 and D_6 in Figure 6 is D_{56}, and is calculated as $h(D_5, D_6)$. The root of the tree is the block digest, and the block signature is the signed block digest. To individually verify a packet, the packet must carry its own authentication information. In this method, the packet must also contain the block signature, the packet position within the block, and the siblings of each node in the packet's path to the root. If D_5 needs to be verified, then so does its entire path to the root. The receiver first computes the digest D'_5 of the packet, and then computes the digests of each of its ancestors in the tree. In this case, those digests would be $D'_{5-6} = h(D'_5, D_6)$, $D'_{5-8} = h(D'_{5-6}, D_{7-8})$, and $D'_{1-8} = h(D_{1-4}, D'_{5-8})$, where D_6, D_{7-8}, and D_{1-4} are contained in the signature of D_5. The receiver then verifies whether D'_{1-8} is equal to the block digest D_{1-8} in the block signature $sign(D_{1-8})$. As with star chaining, verified nodes may be cached, so if another node needs to be verified there are already several nodes along the path to the root that have been verified. This caching greatly speeds up the verification operation if another packet in the same block needs to be verified.

This proposal also introduces several modifications and improvements to the above schemes. These improvements aim to increase the signing and verification speed, and efficiency of the system. Since the experimental part of this thesis feeds off these improvements, their explanation will be saved until the experiments for this thesis are introduced.

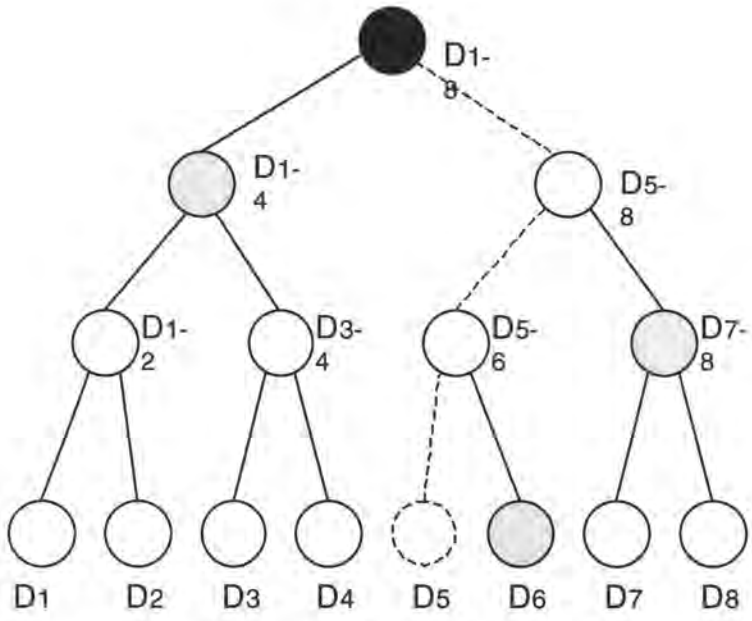

Figure 6: Tree-chaining technique

3.12 Internet Protocol Security

Internet Protocol Security (IPsec) was introduced to provide a standard manner for IP payloads to be authenticated and encrypted. IPsec provides access control, data origin authentication, replay protection, confidentiality, and connectionless integrity by using two protocols: Authentication Header (AH) and Encapsulating Security Payload [14]. IPsec may be implemented in three methods. The first is integration into native IP implementations, and is only viable if access to the IP source code is available. The next option is a Bump-in-the-stack (BITS) implementation. IPsec is implemented underneath an existing protocol stack, between the native IP and the local network drivers. This approach is usually implemented in hosts. The last approach is known as Bump-in-the-wire (BITW), because an outboard cryptographic processor is attached to the system. This may operate like a BITS implementation when used with a host; however, when used with a router, it is used more as a security gateway.

The ESP is designed to provide various security services to both IPv4 and IPv6, and may be applied alone or along with AH. ESP provides confidentiality, data origin authentication, connectionless integrity, and an anti-replay service. ESP operates by

encrypting the static elements of an IP packet, such as the data and TCP header. ESP does not encrypt the IP header, because this must be viewed by the nodes along the packet's path. If ESP's authentication flag is selected, then an ESP authentication value will be computed and placed at the end of the encrypted packet. This value is a variable length field that contains an Integrity Check Value (ICV) computed over the ESP packet. If the sent and computed ICVs match, the packet is deemed valid; otherwise, the packet must be discarded as invalid. The key of ESP is that it not only can authenticate packets, but it also encrypts its member packets.

AH operates in a different manner; its only goal is to authenticate the packet, not encrypt the data. Moreover, AH is able to protect the integrity of some of the IP header and all of the upper level protocol data. AH operates identical to ESP's authentication operation; an ICV is placed at the end of the packet, and this is used by the receiver to verify the authenticity of the packet. The difference of AH is that if there are several IP headers, for example in a tunnel mode, then the AH may create a new IP header field, and protect the inner IP header values with the ICV. AH is also able to protect any payload and IP header of packets that have been encrypted by ESP. In this way, AH and ESP may work together to provide an encrypted and fully authenticated IP packet. However, both of these methods require operations to every packet, and some operations in AH may be rather complex. This is not well suited for a multicast network where a server will be sending multiple packets with different addresses, and therefore IP headers. IPsec has been well established as a security protocol, and warrants use in unicast systems; however, many improvements are needed for it to work efficiently in a multicast system.

4. PROPOSED SOLUTION

The method this thesis will be augmenting is described in [25], *Digital Signatures for Flows and Multicasts*. The first section will continue the description from section 3.10, including a detailed explanation of the algorithms it uses. From there, the improvements suggested will be introduced and explained in detail. Several theoretical and experimental results will be provided to show how modifying this scheme can provide a scheme that will send less bytes across certain multicast networks.

4.1 Existing Method

The essence of the existing proposal is that it creates chaining techniques for signing and verifying multiple packets in a single operation. The basic proposal and its methods have been described in section 3.10. What have not been described yet are the improvements to this scheme that are also introduced in [25]. The first improvement is extended Fiege-Fiat-Shamir (eFFS), which is based on FFS. In the original FFS scheme, with security parameter (k, t), the signer chooses two large primes p and q, then computes a modulus n = pq. The signer then creates k integers $v_1,\ldots,v_k$ and computes $s_1,\ldots,s_k$ as $s^2 = v_i^{-1} \bmod n$. The signing key created is $\{s_1,\ldots,s_k,n\}$ and the verification key is $\{v_1,\ldots,v_k,n\}$.

There are three steps to sign a message m. The first step is to choose t random integers, $r_1,\ldots,r_t$ between 1 and n, and to compute $x_i = r_i^2 \bmod n$ for i = 1, …, t. This creates t integers of size no greater than $|n|$ to be used by the verifying user. The next step is to compute a message digest $h(m, x_1,\ldots,x_t)$. The message digest function h() is publicly known, and the output produced should be at least $k \times t$ bits long. Let $\{b_{ij}\}$ be the first $\{k \times t\}$ bits of the message digest where $i = 1,\ldots,t$ and $j = 1,\ldots,k$. For the third step, the signer calculates $y_i = r_i \times \{s_1^{b_{i1}} \times \ldots \times s_k^{b_{ik}}\} \bmod n$ where $i = 1,\ldots,t$. The signature of a message m is $\{y_i\}$ for $i = 1,\ldots,t$ and $\{b_{ij}\}$ for $i = 1,\ldots,t$ and $j = 1,\ldots,k$. To verify the signature of message m, the verifier calculates $z_i = y_i^2 \times \{v_1^{b_{i1}} \times \ldots \times v_k^{b_{ik}}\} \bmod n$ where $i = 1,\ldots,t$. The signature is

certified as valid if and only if the first $k \times t$ bits of $h(m, z_1, \ldots, z_t)$ are equal to the $\{b_{ij}\}$ received.

The most important aspect of an authentication scheme is its security. Even if a scheme is fast, if it is easily broken it is not useful in most situations. The security level of FFS(*k*,*t*) depends on two different factors. The first factor is the size of modulus *n*; in other words the resulting sizes of *p* and *q*. The second factor is the value of the product *kt*. For both of the factors, the higher its value is the more secure the system is. If different systems have the same modulus value and the same *kt* product, even if their individual *k* and *t* values are different, then the systems have approximately the same security level. In this scheme, the key size is $(k+1) \times |n|$ bits; it consists of *n*, and *k* instances of v_i or s_i, each of which are *n* bits long. The signature size is $t \times |n| + k \times t$ bits. This is because the signature contains *t* values of y_i, each of which is n bits long, and b_{ij}, where *i* is from 1 to *t* and *j* is from 1 to *k*. As can be seen, the signing and verification key sizes depend only on *k*, but the signature size is proportional to *t*. Therefore, to reduce the signature size for a fixed *kt* product, a smaller *t* can be used with a larger *k*. This relationship will prove useful in the next parts of this section.

To speed up the original FFS, three improvements have been suggested in [25]. The first improvement is a smaller verification key. The initial idea was to use the first *k* prime numbers as verification key components to reduce its size. However, the prime number used must satisfy the condition that there exists an integer *s* such that $s^2 = p^{-1} \bmod n$, and not every prime number satisfies this condition. Therefore, the small verification key is created by using the first *k* prime numbers that satisfy the above condition.

The next is improvement is termed the Chinese Remainder Theorem, and it works by reducing the number of modulus operations required. To sign using FFS, $y_i = r_i \times \left\{ s_1^{b_{i1}} \times \ldots \times s_k^{b_{ik}} \right\} \bmod n$ must be computed. According to the Chinese Remainder Theorem, for $n = pq$ a signer can calculate y_i from a_i and b_i using $y_i = \left((a_i - b_i) \times q \times q_p^{-1} + b_i \right) \bmod n$ where $q_p^{-1} = q^{-1} \bmod p$, $a_i = r_i \times \left(s_1^{b_{i1}} \times \ldots \times s_k^{b_{ik}} \right) \bmod p$, and $b_i = r_i \times \left(s_1^{b_{i1}} \times \ldots \times s_k^{b_{ik}} \right) \bmod q$. Therefore, in place of calculating y_i directly from multiplication operations of mod *n*, the signer may first calculate a_i and b_i, which are in mod

p and mod q, and y_i is then calculated from a_i and b_i. Since p and q are much smaller than n, this modulus operation is more efficient, and the signing time is decreased. The verification time cannot be decreased, however, because the knowledge of the factors of n must be known, and only the signer knows these factors.

Precomputation is the last improvement applied to FFS; it uses additional memory to create a more efficient signing operation. For this example, let k=4. The signer calculates $y_i = r_i \times \left\{ s_1^{b_{i1}} \times \ldots \times s_4^{b_{i4}} \right\} \bmod n$ for $i = 1, \ldots, t$. The values $s_1, \ldots, s_4$ remain the same from message to message, and $b_{i1}, \ldots, b_{i4}$ are either one or zero, so the signer can precompute the values (mod n) of every $\{s_1, \ldots, s_4\}$ that is nonempty. To sign a message, the signer only has to multiply that value, mod n, by r_i to obtain y_i. If k is very large, then these precomputations are not practical to precompute. In this case, the subset is partitioned into several smaller subsets that are precomputed individually. If x bits are precomputed in each subset, it is denoted an x-bit precomputation.

Using eFFS, the last major modification in this scheme is adjustable and incremental verification. This is introduced because in a multicast setting, not all receivers are identical, and not all receivers can authenticate in real time at the level they desire. An adjustable verification allows a receiver to choose from several levels of authentication when receiving a message. An incremental verification allows a receiver to return after the fact and increase the security level of any previously authenticated messages. The easiest manner to accomplish this is to use multiple keys of different security levels to sign each packet. This method works, but is inefficient because each packet will have to be signed with every authentication level before the next packet can be signed.

A better method is to use t greater than one, and include $\{x_i\}$ for $i = 2, \ldots, t$ in each of the signatures; this is termed a t-level signature. To verify a message m at security level l of t, the verifier performs the following two steps. First, it calculates $z_i = y_i^2 \times \left(v_i^{b_{i1}} \times \ldots \times v_k^{b_{ik}} \right) \bmod n$ for $i = 1, \ldots, l$. It then verifies that $z_2, \ldots, z_l$ are equal to the appended $x_2, \ldots, x_l$, and that the first $k \times t$ bits of $h(m, z_1, x_2, \ldots, x_t)$ are equal to the $\{b_{ij}\}$ received in the signature. To increase the security level, two more steps are followed. The signer first calculates $z_i = y_i^2 \times \left(v_i^{b_{i1}} \times \ldots \times v_k^{b_{ik}} \right) \bmod n$ for $i = l_1 + 1, \ldots, l_2$, then verifies that

$z_{l_1+1}, \ldots, z_{l_2}$ are equal to $x_{l_1+1}, \ldots, x_{l_2}$. The size of any t-level signature is $kt + (2t-1) \times |n|$ bits, as opposed to $kt + t \times |n|$ for a standard signature using FFS. For example, a signature with a 512-bit modulus and product *kt* of 128 bits has a 1-level signature size of 80 bytes and a 2-level signature size of 208 bytes. The signing time does increase as *t* also increases, but this is still faster than signing a packet once for each authentication level desired.

4.2 MeFFS

The method this thesis proposes takes advantage of multicasting and its properties to create a more efficient authentication scheme for multicast networks, hence a name of Multicast eFFS (MeFFS). If a broadcasting node needs to send packets to multiple levels of receivers, then the above method can be modified to create an optimal scheme. The essence of the new method is to create several multicasting groups if the proper network characteristics are present. If the sender is sending to receivers that require different methods of authentication, instead of sending every level of authentication to each node, it may be more efficient to create a multicast group for each, or a group of, authentication levels. In the above method every authentication level is sent to every receiver, whether it needs it or not. This is useful if the receiver is planning on incrementing the authentication level at a later time. However, in situations where receivers are requesting different authentication levels with no later adjustments, then it is not necessary to use the bandwidth for these extra signatures. This method will take advantage of situations where different sections of a multicast group can be sent only certain authentication levels.

As shown in the previous section, to provide incremental verification a sender will send the packet as normal, but will append the $\{x_i\}$ for $i = 2, \ldots, t$. This means for a 2-level signature, one extra $\{x_i\}$ will be appended, and for a 4-level authentication, three extra $\{x_i\}$ will be appended. Recall that the size of any t-level signature is $kt + (2t-1) \times |n|$ bits. In a frequently used configuration, *n* will be 512 bits and *kt* will be 128 bits. This means that each increase in *t* will add 2*n*, or 1024 bits to the signature size. From the equation above, a 1-level signature is 80 bytes (640 bits) and a 2-level signature is 208 bytes (1664 bits).

Therefore, for each difference in authentication level that is sent to a separate group in this example, 128 bytes (1024 bits) will be saved because one less authentication level will be sent per packet. The key here is that this 128-byte savings may also be propagated throughout the network. In cases where the two levels are completely separated (see Figure 7), the highest savings will be met. As can be seen, for level 1 authentication on the left half of the tree, no extra information will be sent. This is because extra information only needs to be sent for levels of authentication higher than 1. eFFS will still send authentication information for all four levels, because it does not differentiate between levels. For level 4 authentication on the right half of the tree, 384 extra bytes will need to be sent, because three additional x_i will be attached to the messages. MeFFS will send less bytes because it is able to send different authentication information; in this case it will send less information on the L1 half of the tree. The total savings for this example is (384 bytes)*(7 links) = 2688 total bytes.

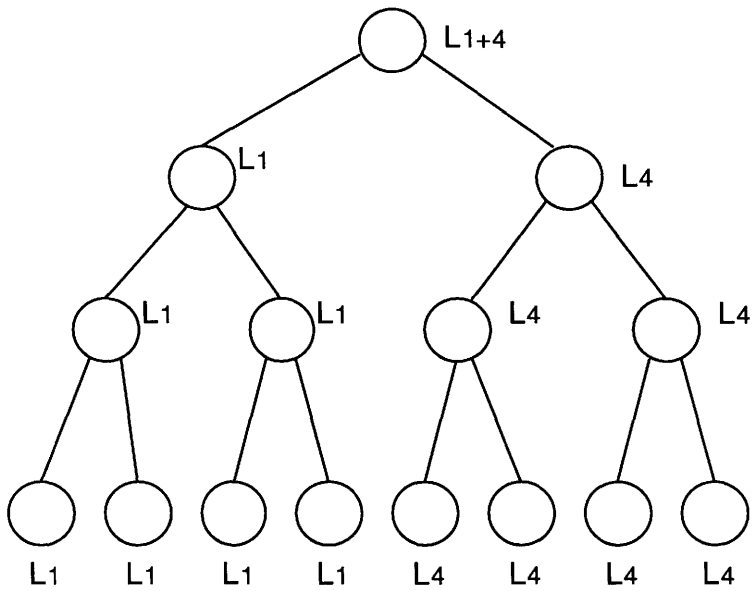

Figure 7: Tree optimized for two levels of authentication

From a theoretical point of view, MeFFS is a very scalable method. If the bandwidth savings discussed above are present on a 15 node network, then a similarly configured network of 150 or 1000 nodes will also send less bytes than eFFS. Moreover, the larger a network becomes, the possible savings are higher because there are more hops on the

network on which to send a smaller amount of data. The processing overhead in MeFFS is slightly higher than eFFS. MeFFS must send packets for each level, and therefore must create the proper packets. This also introduces an overhead of sending multiple packets for each message; this disadvantage will be discussed later in the thesis. MeFFS and eFFS also have an approximately equal ease of deployability. The nodes must let the root of the tree know which level it desires, and if its desired level changes. To deploy MeFFS, receivers must be able to alert senders in this manner, but this is not an extremely complex task.

The purpose of the experiments is to determine in which settings, including randomly selected average settings, do the multicast group savings offset any extra packets being sent when multiple levels of receivers are on the same branch. Moreover, if there are networks which can't be separated optimally, is there a method which will still create a lower load on the network? These questions will be answered in the following section.

5. EXPERIMENTS

The experiments for this thesis are run using Network Simulator (ns), a powerful simulation software developed at the University of California, Berkeley. The objective of these experiments is to randomly generate several networks of various sizes and configurations, and to determine which parameters create a multicast authentication scheme more efficient than the existing situation. This section will describe how the experiments are setup, what the results are, and how the results may be used to lessen a network's load.

5.1 Experiment Setup

The first step in the process is to create a random network of a given size. The random network is created using a random number generator provided by ns. The random number generator is seeded at 0, making it only a statistically random network; this is sufficient for the purposes of these experiment. The next problem is to assign authentication levels to each of the nodes in a random manner. This is done in a similar manner as creating the network. A list of numbers is created containing the values 0 to i, where i is the total number of levels present in the network. After this list is created, its members are sequentially assigned as authentication levels to nodes of the network.

The network this creates is randomly connected, and several redundant links are created. To properly determine the amount of bytes sent throughout the network, the multicast tree within this network must be found. Therefore, before sending or processing anything within the program, the network must be parsed to determine the proper multicast tree. Figures 8 and 9 show an example network generated by ns, and its corresponding multicast tree, respectively. Finding the multicast tree is carried out by starting at each node in the network and tracing its path back to the root. A list keeping track of these paths is created as the network is traversed. After the list is created, any connections in the multicast network that are not contained in the list can de discounted, and an accurate model of the network is then created.

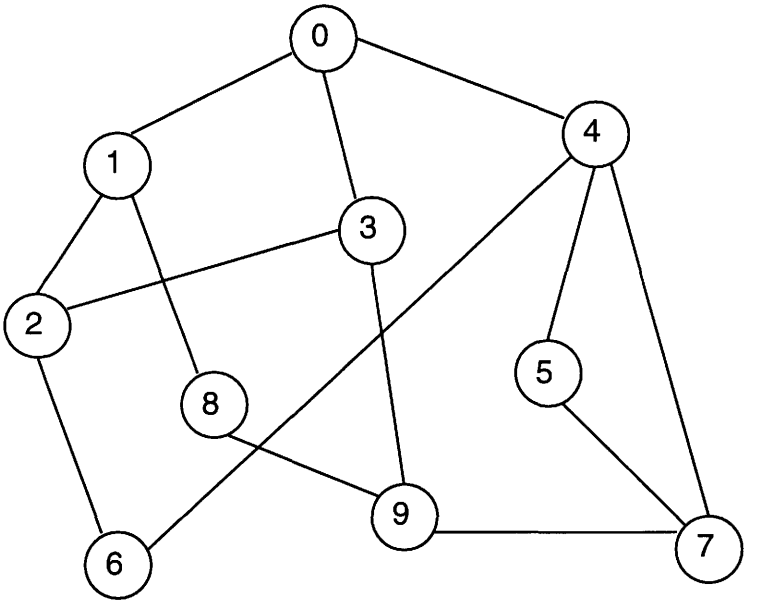

Figure 8: Randomly generated network

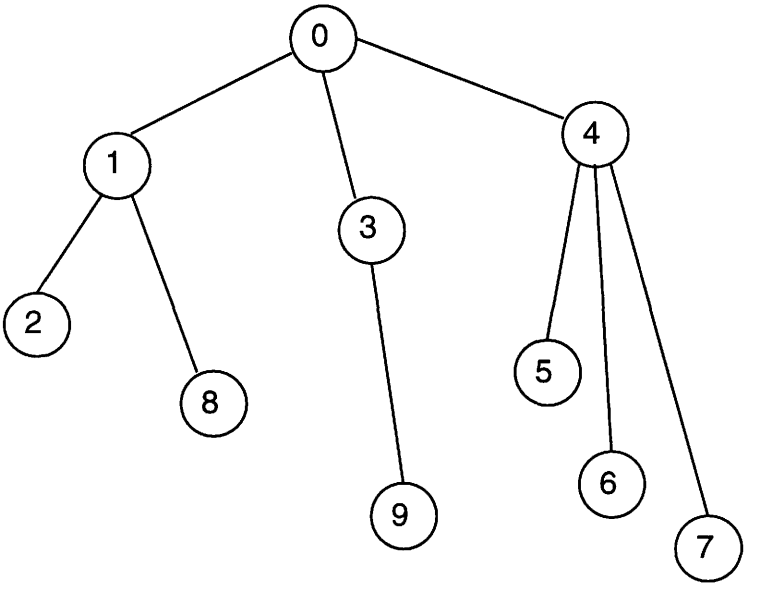

Figure 9: Multicast tree of Figure 8

The next step is to determine, for both the old and new methods, how many bytes will be flowing across a particular path pr message signed by the sender. While the network is being traversed to find the multicast tree and a count of the number of hops in the path from a particular node to the root is taken, the levels of authentication across those legs are also

recorded. If the paths from Node A and Node B to the root merge at any point in the tree, then noting the levels involved becomes very important. If one path is serving two or more nodes, then it is possible that more than one packet will flow across that segment for each packet presented to the sender for signing. This is the case for the new method proposed in this thesis, MeFFS. If Nodes A and B use the same authentication level, then the hops of the path they share should only be counted once. However, if Node A desires level 1 and Node B desires level 4, then the path should be counted as two distinct paths, because one packet will be sent signed at level 1, and another packet will be sent signed at level 4. However, in the existing eFFS method, each hop is counted only once, no matter how many nodes are using that leg, because the same amount of information is being sent across that hop. The information in each packet includes the authentication information for every level, so one packet is sent regardless of the requirements for the lower nodes of the tree.

The basis of the experiment is to determine if there are situations where it is more efficient to use MeFFS than the existing eFFS, and when these situations are present in multicast networks. It may not be the case that every multicast network will be suitable for applying separate multicast groups. However, if it is easy to determine what types of networks will benefit from this method, or how a network can be adjusted and routed to become more efficient, then the new method definitely warrants further study and implementation. A simple example of how to determine the total network load is presented in Figures 10 and 11. Figure 10 shows a network with nodes of various levels, and the packet size flowing across each of the legs. For the experiments in this thesis, an average packet size of 750 bytes is assumed. The maximum Ethernet packet is approximately 1500 bytes, and a level 4 signature is 464 bytes. To keep packets from being fragmented and reassembled, 750 bytes is a good approximation for a packet size before it is signed by the sender.

Let a 2-level authentication scheme be the basis for this example, where kt is 128 bits and n is 512 bits. In eFFS, each 2-level packet has a signature of $kt + (2t-1) \times |n|$ bits, or 208 bytes. The total number of signature bytes flowing across this network is $6 \times 208 = 1248$ bytes. With MeFFS, packets intended for level 1 receivers are of size 80 bytes, and packets for level 1 or level 2 receivers are 208 bytes, as explained in section 4.2. The total number of signature bytes flowing across this network is $3 \times 80 + 3 \times 208 = 864$

bytes. This means the total number of bytes sent for signatures in the new method is less than that of the old method. However, since the new method has one link in which two packets are sent, this may offset the savings. The total of bytes sent across the network with MeFFS is 5364, while eFFS sends 5748 bytes. This scenario is definitely suited for the new method; further experiments and evaluation will determine which network settings are good for using this newly proposed method.

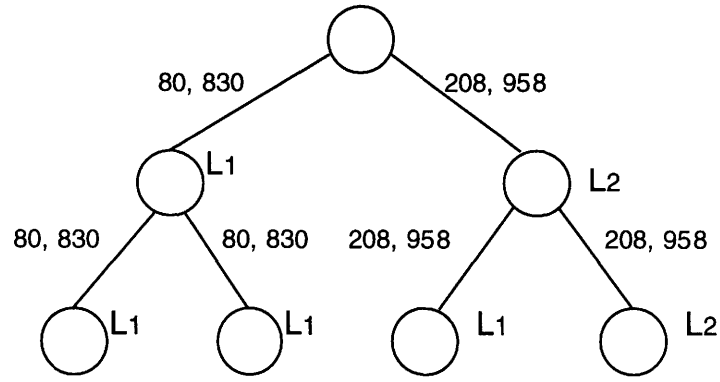

Figure 10: Signature, packet size for new method

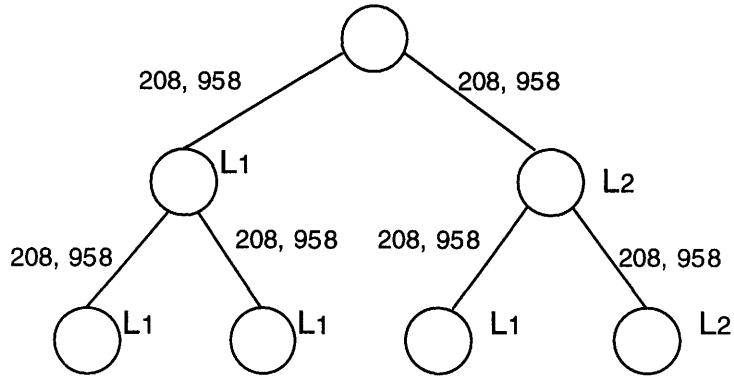

Figure 11: Signature, packet size for old method

5.2 Experiments

Several parameters are introduced to make these experiments as useful as possible. The first parameter is the network size, or total number of active and inactive nodes in the network. These experiments are run on networks of 25, 100, and 250 nodes. The number of levels that can be assigned randomly to nodes is also varied. Simulations are run on networks

with 1, 2, and 4 levels. Nodes may also be assigned to level 0, which means it is not actively receiving packets, but may act as a forwarding node for other parts of the tree. With 1 level, every method will have the same results, so it may be used as a common starting point. Levels 2 and 4 are chosen because these are the level values for which there is sample data for the eFFS method. These levels are also the most likely to be used; a 4-level signature is already 464 bytes, any more will place an excessive overhead on each packet. With these parameters, each test is run three times with different random variables for both network creation and level assignment. For the graphs shown, the data points are the average of the three runs.

Figures 12, 13, and 14 show the total number of bytes on an eFFS and MeFFS network of 25, 100, and 250 nodes respectively. As can be seen from the graphs, the MeFFS method increases the total amount of bytes sent on a randomly created network. The reason this happens is because in a network in which the receivers have randomly assigned levels, these levels will be widely scattered. This does not create an optimal solution, because in MeFFS, if there is a path with receivers of level 1, 2, and 3, then each of those packets will need to flow across the shared path. The next step is to determine how to improve this.

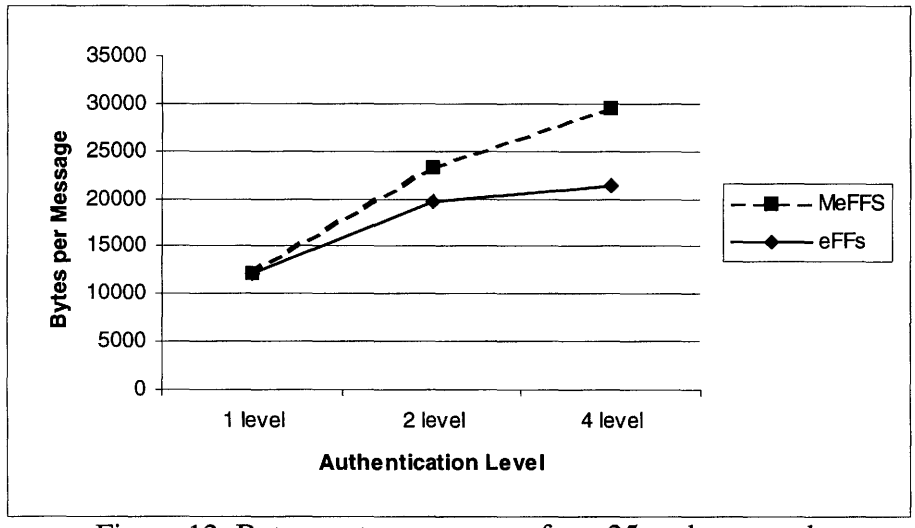

Figure 12: Bytes sent per message for a 25 node network

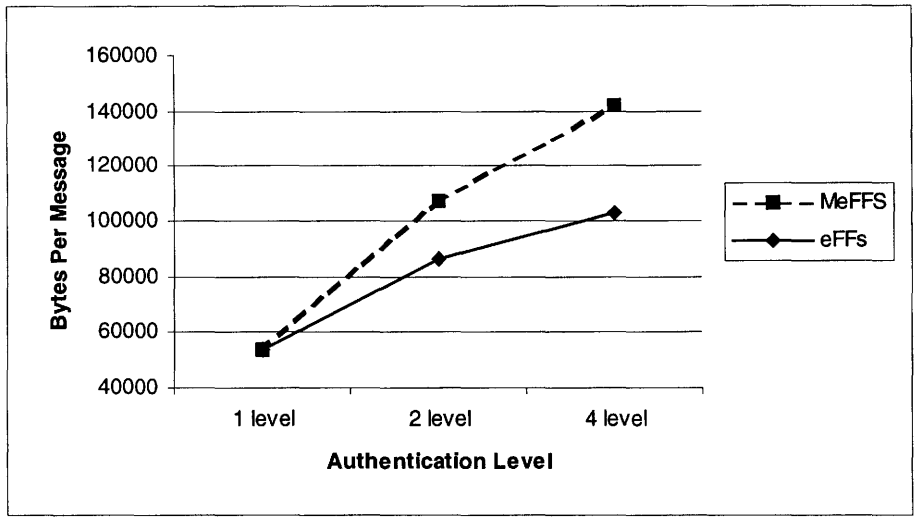

Figure 13: Bytes sent per message for a 100 node network

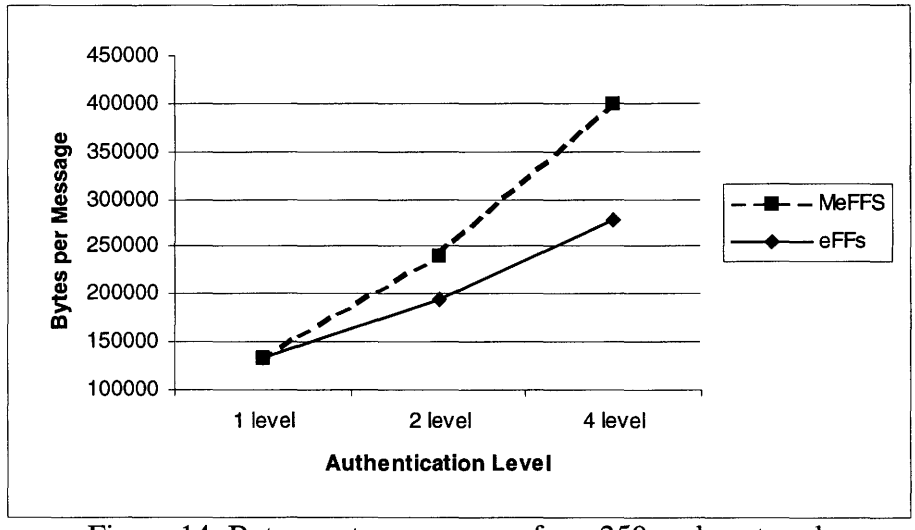

Figure 14: Bytes sent per message for a 250 node network

There are three separate ways this path sharing can be overcome, two of which are very viable. The first is to have packets that are very short before they are signed. This would mean that any links that are shared would have a very small redundancy level, because the shared packet payload would be small and the signature size would be the bulk of the data. However, this is not a realistic situation, because once there is a system that sends more authentication bits than data bits the efficiency is drastically reduced.

The second option is a much more likely, and existing, method. As explained, if there are less common links in a network between different levels of authentication, then the total

savings in the network is increased. If only packets authenticated at level 1 travel on one branch, and packets of level 4 travel on another branch, then MeFFS will be an optimal solution. Therefore, if a network can be arranged such that this situation exists, MeFFS will be very well suited to that type of network. One such example of this occurring naturally is where a multicasting sender is sending to several corporate LANs and to a cellular network. With the advent of universal and packetized cellular networks, end users are likely to desire a faster packet rate at the expense of a lower authentication level. There is also the middle segment of home users on modem networks. These users may opt for a slightly higher level of authentication than cellular networks, but still desire lower authentication levels than corporate users connected via high-speed links. Figures 15 and 16 will demonstrate this.

Figure 15 shows an actual multicast network used in the experiments for this thesis. The numbers inside the nodes represent the levels of the particular node. With this network, using eFFS 24,280 bytes are sent across the entire network for each message signed by the signer. With MeFFS, this increases to 30,152 bytes. However, Figure 16 shows the same network, with the levels of only two nodes changed, shown in bold and marked by arrows. One node is changed from level four to level three, and another from level four to level two. Since the same packets are sent for any signature level in eFFS, the total byte count is still 24,280 bytes. However, this small change has lowered the total sent value for MeFFS to only 23,826 bytes. This means for every message signed by the sender, 454 bytes will be saved using the new method. This is a small value; however, this example is on a very small scale. First, only two nodes have been changed. If nodes can be grouped together according to authentication level, or they are naturally grouped as in a LAN versus mobile environment, then a much higher savings will be realized. Second, this example is for a network with only twenty-five nodes. The larger a network is, if it is configured correctly, the more the savings will multiply.

As an example, take a multicast network of 1000 nodes, where 500 nodes request level one and 500 nodes request level four. Moreover, the different levels are on separate branches, separating the users optimally. If this is modeled as a binary tree, then there will be 500 links of level 1 authentication, and 500 links of level 4 authentication. With eFFS the same packet size of 750 bytes data + 464 bytes signature = 1214 bytes total will be used.

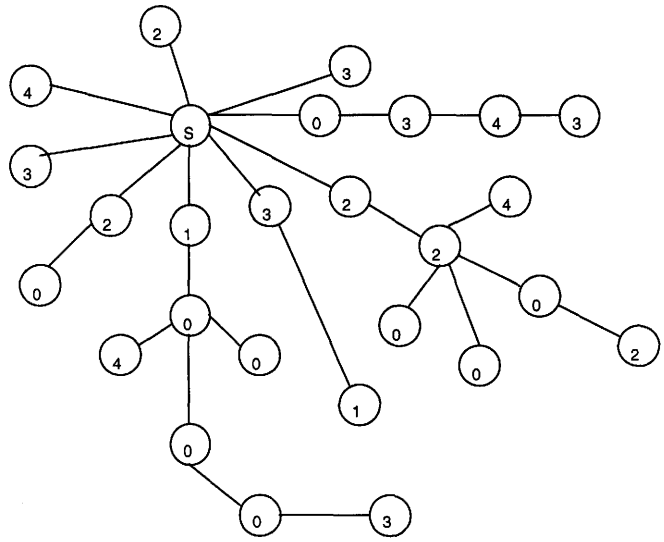

Figure 15: 25 node, 4 level multicast network

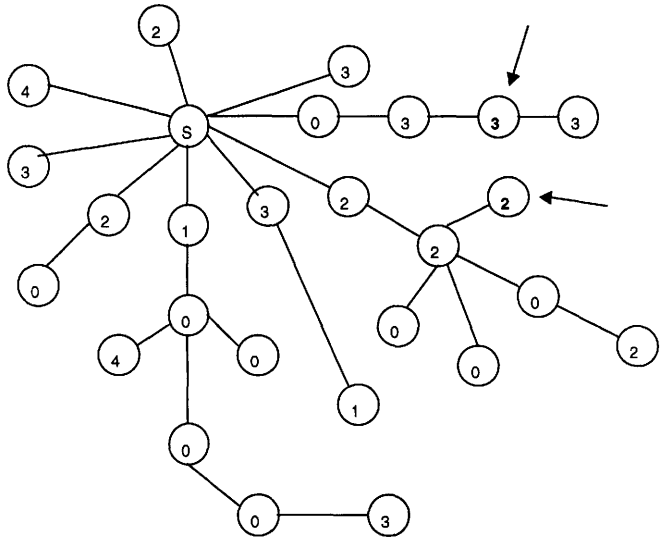

Figure 16: Modified 25 node, 4 level multicast network

This is a total of $1214 \times 1000 = 1214000$ bytes of data, or 1186kB. MeFFS will send out 500 level one packets at 750 bytes data + 80 bytes signature = 830 total bytes. It will also send out 500 level four packets at 750 bytes data + 464 bytes data = 1214 bytes. Across this network, $(830 \times 500) + (1214 \times 500) = 1022000$ bytes or 998KB will be sent for each message

signed. This is a savings of 188kB per message, and if 1000 messages are signed per second, this is a total savings of 184MB of data per second. It is not likely a network will be separated so perfectly, but this gives an example of the power of MeFFS in the proper conditions. Future experiments on routing algorithms could determine how easily it is to create multicast networks that are separated by level, or at least partial toward like level branches, instead of a randomly connected network.

The last method for optimizing MeFFS is also created to lessen multiple payload data being sent on the same links. This solution assumes a relatively powerful sending processor. Instead of sending entire signed packets to each receiver, it is possible to send the packets and the authentication data separately. This is still secure, because if the pure data packet or the signature packet is modified in any way, then the signature will be invalid and the packet may still be rejected. It will be required to associate each packet with its signature, but this can be accomplished with a simple ID tag. This tag can be present in the data packet, signed into the signature packet, and appended to the signature packet. It will be appended to the signature packet for receivers to know which signatures to use for verification, and inside the signature to verify the ID tag has not been modified. Figures 17, 18, and 19 compared the eFFS method with this modified MeFFS method for the same networks of 25, 100, and 250 nodes as the above figures. As can be seen in the figures, this modified method is more efficient than the eFFS method in many experiments. This is not the case 100% of the time, but these experiments are using a randomly created multicast network. This system may also be combined with less a trivial routing algorithm to create a more optimized network for the modified MeFFS.

From the experiments and explanations in this section, it can be seen that it is difficult to create a completely optimized solution where there are multicast networks with different levels of receivers scattered throughout the network. However, there are several ways to improve on the existing methods. Some of them work best with a multicast tree modified so levels are separated by branches, although there are many situations where the tree is separated by the nature of the users. In these situations, the MeFFS and modified MeFFS are well suited as a signing method, and warrant further study and experimentation.

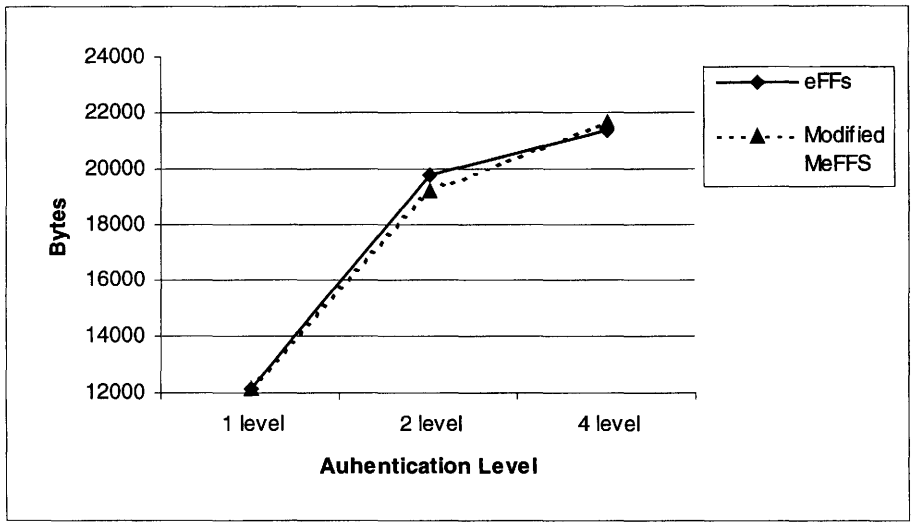

Figure 17: Bytes sent per message for 25 nodes

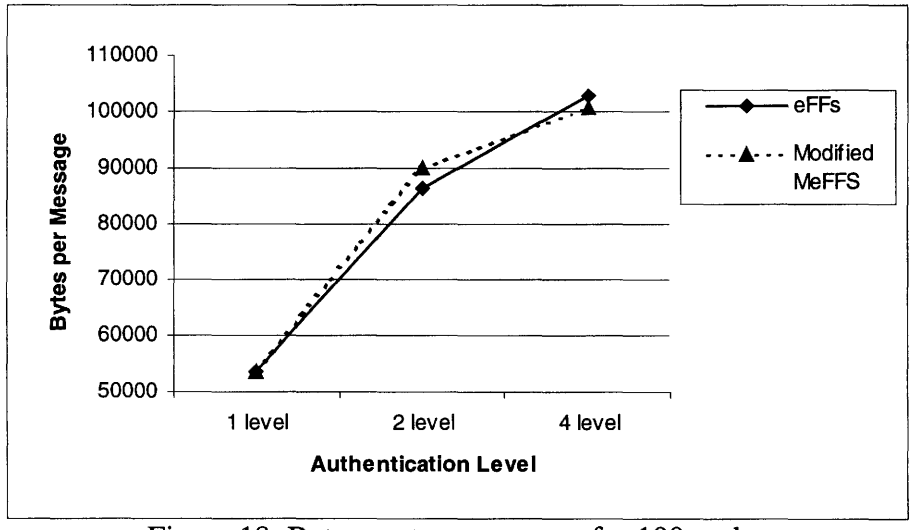

Figure 18: Bytes sent per message for 100 nodes

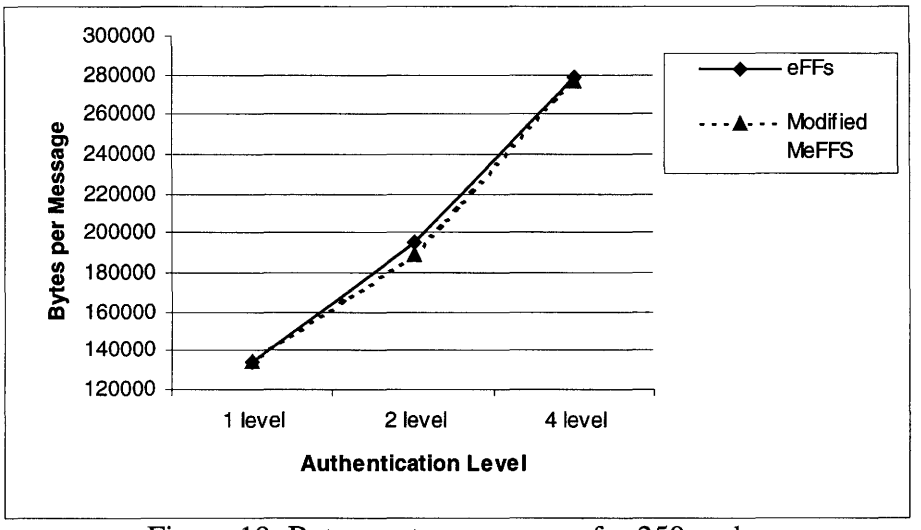

Figure 19: Bytes sent per message for 250 nodes

6. CONCLUSIONS

Since the advent of communication, several issues have been constantly studied to improve its capabilities and integrity. One of these issues is how to effectively send messages to a large number of people efficiently; broadcasting and multicasting have been introduced and explored as solutions to this problem. Another issue that has always been important in remote communication is security. Users need to be sure that their message is being sent to the correct user, in the original format, and in some cases that no other users can see the contents of the message. The scope of this thesis is on source and data authentication, not necessarily encryption. This means the users are concerned about the message integrity and the knowledge of who sent the message. Since both multicasting and authentication have been of high priority in communications, it is no surprise that attempts have been made to combine the two ideas into a common, efficient system.

Several methods are proposed for multicast authentication, and these methods are measured by several metrics. These methods must take into account different capabilities of receivers, and be able to scale to a large number of receivers. They must also be resilient against collusion of users within and outside the network, and be efficient in use. Two basic schemes are available for multicast authentication. They are public key networks, and hashing and signing algorithms. Public key signing is simple, but it is computationally expensive and creates large signatures. The second method is MACs, which are more efficient and create smaller signature and key sizes. Several systems are based on this method, many of which are technically feasible. One improvement made to this is to use UOWHFs, or one-time hash functions. These are less expensive to process, and if the functions are used only once, are just as stable as more conservative hash functions. Other improvements include an on-line/ off-line signature scheme, where the slow algorithms are run before the message arrives, and faster algorithms are processed on the message when it arrives. Many systems are also based on stream signing. This is more efficient than signing each message individually, but it requires a network with an error rate of zero, which is very unlikely in current networks.

The solution proposed is an improvement to a signing method used in a flow-based signature scheme. The signature scheme in use, eFFS, allows packets to be signed such that

receivers can authenticate the packets at their desired levels. This is done by including information for all of the potential levels, usually 1, 2 or 4. This is efficient compared to existing methods, but improvements still can be made. Multicast eFFS allows the network to be split into different groups according to the receivers' desired levels. For example, if one part of a network desires level 1 authentication and another part desires level 4 authentication, then only that authentication level will be sent to that part of the network. This saves on unnecessary signature levels being sent throughout the network. However, if the network is widely scattered with levels of different receivers, then this method is not as efficient as eFFS. Several options are available for alleviating this constraint.

The first option is to send much smaller packets; this creates less duplicate data but at a sacrifice of sending packets that are too small to be useful. The second option is to create, or use, networks that are created in tune with the level of the receivers. Only minor changes in a network's structure will allow the MeFFS method to be more efficient, and some networks are already created where different levels are on different branches. The last option is to send the data to be authenticated and the signatures in separate packets. In this way, the data goes to every user, but the authentication packets only go to the receivers who require it. This creates a higher overhead from sending multiple packets, but with a powerful sender, which is usually the case, this tradeoff will create a less loaded network. When this method is combined with networks that are structured at least somewhat according to authentication level, a powerful and efficient multicasting authentication method is created.

In summary, this thesis:

- Introduces communication, multicasting, and multicast security
- Explains multicast authentication methods, stemming from
    - Public key systems
    - Message Authentication Codes
    - Digital Stream Signing
- Proposes an improvement that splits users into multicast groups, allowing efficient authentication at multiple levels
- Identifies two modifications to the proposed scheme: routing by authentication and sending signatures separate from the packet

Future work may help the MeFFS proposal in two areas:

- Create a multicast routing scheme with authentication level as a parameter.
- Create a signature scheme that requires separate, not additional information to verify at incrementally higher authentication levels.

The routing scheme may determine the complexity of arranging networks in a manner optimal for MeFFS type of routing, with levels having their own branch of the tree. A mathematical analysis of signature schemes may determine if it is possible to create a method similar to MeFFS but does not require extra information for all levels up to the desired authentication level. These studies would continue the work of MeFFS and possibly create a more efficient and still highly secure multicast authentication system.

REFERENCES

[1] M. Bellare, P. Rogoway, "Collision-Resistant Hashing: Towards Making UOWHFs Practical", *Advances in Cryptology- CRYPTO '97*, LNCS vol. 1294, Springer-Verlag, 1997, pp. 470-484.

[2] J. Black, S. Halevi, H. Krawczyk, T. Krovetz, P. Rogaway, "UMAC: Fast and Secure Message Authentication", *Proc. CRYPTO '99*, LNCS vol. 1666, Springer-Verlag, Aug. 1999, pp. 216-233.

[3] D. Bleichenbacher, U. M. Maurer, "Optimal Tree-based One-time Digital Signature Schemes", *Proc. STACS '96*, LNCS vol. 1046, Springer-Verlag, 1996, pp. 145-158.

[4] R. Canetti, J. Garay, G. Itkis, D. Micciancio, M. Naor, B. Pinkas, "Multicast Security: A Taxonomy and Some Efficient Constructions", *IEEE INFOCOM,* 1999.

[5] D. Catalano, R. Gennaro, "New Efficient and Secure Protocols for Verifiable Signature Sharing and Other Applications", *Proc. CRYPTO'98*, LNCS vol. 1642, Springer-Verlag, 1998, pp. 105-120.

[6] I. Chang, R. Engel, D. Kandlur, D. Pendarakis, D. Saha, "A Toolkit for Secure Internet Multicast", *IBM Research manuscript*, 1998.

[7] S. E. Deering, "Host Groups: A Multicast Extension to the Internet Protocol", *RFC 966*, Dec. 1985.

[8] S. Even, O. Goldreich, and S. Micali, "On-Line/ Off-Line Digital Signatures", *J. Cryptology*, vol. 9, no. 1, 1996, pp. 35-67.

[9] R. Gennaro, P. Rohatgi, "How to Sign Digital Streams", *Proc. CRYPTO 97*, pp. 180-197, Aug. 1997, Santa Barbara, CA.

[10] S. Halevi, H. Krawczyk, "MMH: Software Authentication in the Gbit/second Rates", *Proc. 4th Workshop on Fast Software Encryption*, LNCS vol. 1267, Springer-Verlag, 1997, pp. 172-189.

[11] N. Haller, R. Atkinson, "On Internet Authentication", *RFC 1704*, Oct. 1994.

[12] S. Kent, R. Atkinson, "IP Authentication Header", *RFC 2402*, Nov. 1998.

[13] S. Kent, R. Atkinson, "IP Encapsulating Security Payload (ESP)", *RFC 2406*, Nov. 1998.

[14] S. Kent, R. Atkinson, "Security Architecture for the Internet Protocol", *RFC 2401*, Nov. 1998.

[15] H. Krawczyk, M. Bellare, R. Canetti, "HMAC: Keyed-Hashing for Message Authentication", *RFC 2104*, Feb. 1997.

[16] C. Madson, R. Glenn, "The Use of HMAC-MD5-96 within ESP and AH", *RFC 2403*, Nov. 1998.

[17] C. Madson, R. Glenn, "The Use of HMAC-SHA-1-96 within ESP and AH", *RFC 2404*, Nov. 1998.

[18] S. Mittra, T.Woo, "A Flow-Based Approach to Datagram Security", *Proc. ACM SIGCOMM*, Cannes, France, 1997.

[19] M. J. Moyer, J. R. Rao, and Pankaj Rohatgi. "A Survey of Security Issues in Multicast Communications." *IEEE Network Magazine*, Nov./Dec. 1999.

[20] B. Quinn, K. Almeroth, "IP Multicast Applications: Challenges and Solutions", *IETF Internet Draft <draft-ietf-mboned-mcast-apps-01.txt>*, June 1999.

[21] P. Rohatgi, "A Compact and Fast Hybrid Signature Scheme for Multicast Packet Authentication", *Proc. 6th ACM Conference on Computer and Communications Security*, Singapore, Nov. 1999.

[22] H. Schulzrinne, S. Casner, R. Frederick, V. Jacobson, "RTP: A Trasnport Protocol for Real-Time Applications", *RFC 1889*, Jan. 1996.

[23] A. Shacham, R. Monsour, R. Pereira, M. Thomas, "IP Payload Compresiion Protocol (IPComp)", *RFC 2393*, Dec. 1998.

[24] R. Thayer, N. Doraswamy, R. Glenn, "IP Security Document Roadmap", *RFC 2411*, Nov. 1998.

[25] C. K. Wong, "Digital Signatures for Flows and Multicasts", *IEEE/ACM Transactions on Networking*, vol. 7, no. 4, Aug. 1999.

[26] K. Zhang, "Efficient Protocols for Signing Routing Messages", *Proc. Symposium on Network and Distributed Systems Security*, San Diego, California, 1998.